Cortex-M3/IAR

scmRTOS

ОПЕРАЦИОННАЯ СИСТЕМА РЕАЛЬНОГО ВРЕМЕНИ

для однокристальных микроконтроллеров

Version 5

2003-2015

Общие сведения

Cortex-M3 — это новое 32-разрядное ARM RISC ядро с гарвардской архитектурой. Микроконтроллеры на основе этого ядра выпускается многими фирмами (ST Microelectronics, NXP, TI и др.), что позволяет разработчику выбрать наиболее подходящий микроконтроллер. В отличие от прежних ядер ARM, в ядре **Cortex-M3** стандартизовано не только ЦПУ, но и контроллер прерываний, системный таймер и карта памяти. Это позволяет использовать данный порт на контроллере любого производителя практически без изменений. Ядро разработано с учётом возможного применения операционных систем, и потому идеально подходит для ***scmRTOS***.

Данный порт предназначен для использования совместно с программным пакетом **EWARM** фирмы **IAR Systems**.

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

Объекты портирования

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

Макросы

Название	Значение[1]
`INLINE`	`_Pragma("inline=forced") inline`

[1] Если значение макроса пусто, то для обозначения этого используется тег <None>.

OS_PROCESS	**<None>**
OS_INTERRUPT	**<None>**
DUMMY_INSTR()	**__no_operation()**
SYS_TIMER_CRIT_SECT()	**TCritSect cs**
SEPARATE_RETURN_STACK	**0**
ENABLE_NESTED_INTERRUPTS	**<None>**

Поскольку **Cortex-M3** поддерживает вложенные прерывания на аппаратном уровне, макрос **SYS_TIMER_CRIT_SECT()** содержит создание объекта - критической секции. Соответственно, макрос **ENABLE_NESTED_INTERRUPTS** пуст. Этот макрос используется в обработчике прерываний системного таймера, если вложенные прерывания в обработчике системного таймера разрешены (конфигурационный макрос **scmRTOS_SYSTIMER_NEST_INTS_ENABLE == 1**).

Порт не поддерживает отдельный стек для адресов возвратов, поэтому макрос **SEPARATE_RETURN_STACK** равен нулю.

Псевдонимы типов

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

Пользовательские типы

Класс-«обёртка» критической секции – см «**Листинг 1 – 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}      INLINE TCritSect ()
{5}          : StatusReg(__get_interrupt_state()) { __disable_interrupt(); }
{6}      INLINE ~TCritSect() { __set_interrupt_state(StatusReg); }
{7}
{8}  private:
{9}      status_reg_t StatusReg;
{10} };
```

Листинг 1 – `TCritSect`

Класс-«обёртка» **`TISRW`** предназначен для упрощения определения обработчиков прерываний, в которых используются сервисы ОС, см «**Листинг 2 – 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}         TCritSect cs;
{12}         Kernel.ISR_NestCount++;
{13}     }
{14}     //-------------------------------------------------------
{15}     INLINE void isr_exit()
{16}     {
{17}         TCritSect cs;
{18}         if(--Kernel.ISR_NestCount) return;
{19}         Kernel.sched_isr();
{20}     }
{21}     //-------------------------------------------------------
{22} };
```

Листинг 2 – `TISRW`

Использование: в обработчике прерываний объект этого класса должен быть объявлен до первого использования любого средства межпроцессного взаимодействия.

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 **Cortex-M3/IAR** использует отдельный стек для прерываний, то есть при входе в обработчик прерывания происходит переключение на отдельный стек. Такой

подход даёт экономию стеков процессов, т. к. в этом случае не нужно в стеках процессов резервировать пространство для работы обработчиков прерываний. В качестве области памяти, выделенной под стек прерываний, используется память, которая была стеком до старта ОС. Реализация этой возможности выполняется аппаратно ядром **Cortex-M3**.

Второй вариант — когда прерывания используют стеки процессов — не реализован в порте. Поэтому **TISRW_SS** и **TISRW** являются синонимами:

```
{1}  #define TISRW_SS TISRW
```

Листинг 3 – TISRW_SS

Системный таймер

Поскольку в спецификацию ядра **Cortex-M3** включен, в том числе и системный таймер **SysTick**, то его настройка выполняется портом. На уровне приложения пользователь определяет только частоту клока, подаваемого на таймер, и желаемую частоту таймерного прерывания. Это делается макросами **SYSTICKFREQ** и **SYSTICKINTRATE** в файле **scmRTOS_TARGET_CFG.h**:

```
{1}  #define SYSTICKFREQ     72000000
{2}  #define SYSTICKINTRATE  1000
```

Листинг 4 – Задание частоты системного таймера

Порядок приоритетов

Порт **Cortex-M3/IAR** поддерживает как прямой, так и обратный порядок приоритетов процессов. Но в силу того, что ядро имеет аппаратные средства поиска первого ненулевого бита в двоичном слове, то предпочтительным (в плане быстродействия и размера кода) является обратный порядок приоритетов, т.е. при конфигурировании системы (файл **scmRTOS_CONFIG.h**) необходимо указать:

```
{1}  #define scmRTOS_PRIORITY_ORDER  1
```

Листинг 5 – Задание порядка приоритетов

Передача управления на основе программного прерывания

Это единственный предусмотренный в порте вариант, поскольку ядро **Cortex-M3** имеет специальное прерывание для этого, и вариант с прямой передачей управления не имеет никаких преимуществ. В порте определёна функция обработки прерываний **`PendSV_Handler()`**, реализованная на ассемблере, которая и 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 6 – scmRTOS_CONFIG.h**».

```
{1}  #ifndef __IAR_SYSTEMS_ASM__
{2}  typedef uint16_t      timeout_t;
{3}  typedef uint_fast32_t tick_count_t;
{4}  #endif // __IAR_SYSTEMS_ASM__
{5}
{6}  #include <stdint.h>
{7}
{8}  #define scmRTOS_PROCESS_COUNT                    3
{9}  #define scmRTOS_SYSTIMER_NEST_INTS_ENABLE        1
{10} #define scmRTOS_SYSTEM_TICKS_ENABLE              1
{11} #define scmRTOS_SYSTIMER_HOOK_ENABLE             1
{12} #define scmRTOS_IDLE_HOOK_ENABLE                 1
{13} #define scmRTOS_IDLE_PROCESS_STACK_SIZE      (50 * sizeof(stack_item_t))
{14} #define scmRTOS_PRIORITY_ORDER                   1
{15} #define scmRTOS_CONTEXT_SWITCH_USER_HOOK_ENABLE  0
{16} #define scmRTOS_DEBUG_ENABLE                     0
{17} #define scmRTOS_PROCESS_RESTART_ENABLE           0
```

Листинг 6 – scmRTOS_CONFIG.h

Вышеприведённый файл определяет два псевдонима встроенных типов – для переменных тайм-аутов {2} и для счётчика тиков системного таймера {3}, число

[1] Только значимая часть, без комментариев, «шапок», code guard'ов и прочего.

пользовательских процессов в количестве 3 {8}, разрешает вложенные прерывания в обработчике прерываний системного таймера {9}, разрешает функцию системного времени – счётчик тиков системного таймера {10}, разрешает пользовательские хуки системного таймера и фонового процесса системы (**IdleProc**) {11}, {12}, а пользовательский хук при переключении контекстов не разрешён {15}, порядок следования приоритетов – обратный (**prIDLE** равно **0**, **pr3** – **1, pr2 – 2** и т. д.){14}. Отладочные средства отключены {16}, рестарт процессов отключен {17}. Также указано подключение заголовочного файла с объявлениями стандартных целочисленных типов {6}.

Файл **scmRTOS_TARGET_CFG.h** содержит код ОС, зависящий от требований конкретного проекта. Его содержимое – см «**Листинг 7 – scmRTOS_TARGET_CFG.h**».

```
{1}  // Define SysTick clock frequency and its interrupt rate in Hz.
{2}  #define SYSTICKFREQ     72000000
{3}  #define SYSTICKINTRATE  1000
{4}
{5}  //-----------------------------------------------------------------------
{6}  // Definitions for some processor registers in order to not include
{7}  // specific header file for various Cortex-M3 processor derivatives.
{8}  #define CPU_ICSR            ( ( volatile uint32_t *) 0xE000ED04 )
{9}  #define CPU_SYSTICKCSR      ( ( volatile uint32_t *) 0xE000E010 )
{10} #define CPU_SYSTICKCSR_EINT 0x02
{11}
{12} #ifndef __IAR_SYSTEMS_ASM__
{13} //-----------------------------------------------------------------------
{14} //
{15} //       System Timer stuff
{16} //
{17} //
{18} namespace OS
{19} {
{20} // SysTick interrupt handler.
{21} extern "C" void SysTick_Handler();
{22} }
{23}
{24} #define LOCK_SYSTEM_TIMER()    (*CPU_SYSTICKCSR &= ~CPU_SYSTICKCSR_EINT)
{25} #define UNLOCK_SYSTEM_TIMER()  (*CPU_SYSTICKCSR |=  CPU_SYSTICKCSR_EINT)
{26}
{27} //-----------------------------------------------------------------------
{28} //
{29} //       Context Switch ISR stuff
{30} //
{31} //
{32} namespace OS
{33} {
{34} #if scmRTOS_IDLE_HOOK_ENABLE == 1
{35}     void idle_process_user_hook();
{36} #endif
{37}
{38} #if scmRTOS_CONTEXT_SWITCH_SCHEME == 1
{39}
{40}     INLINE void raise_context_switch() { *CPU_ICSR |= 0x10000000; }
{41}
{42}     #define ENABLE_NESTED_INTERRUPTS()
{43}
{44}     #if scmRTOS_SYSTIMER_NEST_INTS_ENABLE == 0
{45}         #define DISABLE_NESTED_INTERRUPTS() TCritSect cs
{46}     #else
{47}         #define DISABLE_NESTED_INTERRUPTS()
{48}     #endif
{49}
{50} #else
{51}     #error "Cortex-M3 port supports software interrupt switch method only!"
{52}
{53} #endif // scmRTOS_CONTEXT_SWITCH_SCHEME
{54}
{55} }
{56} //-----------------------------------------------------------------------------
{57} #endif // __IAR_SYSTEMS_ASM__
```

Листинг 7 – scmRTOS_TARGET_CFG.h

В начале файла задаются частота клока системного таймера {2} и желаемая частота таймерного прерывания {3}. Настройка клока для системного таймера определяется конкретным микроконтроллером с ядром **Cortex-M3** и должна быть

выполнена пользователем до запуска ***scmRTOS***. Затем идут несколько определений регистров **Cortex-M3** {8}{9}{10}. Далее определены два макроса, которые управляют разрешением прерываний системного таймера путём манипуляции соответствующим битом разрешения прерывания системного таймера {24}{25}. Для варианта передачи управления с помощью программного прерывания требуется определить функцию **`raise_context_switch()`** {40}, которая активизирует соответствующее прерывание.

Для включения и отключения вложенных прерываний в обработчике прерывания системного таймера определёны специальные макросы. Поскольку у **Cortex-M3** вложенные прерывания включены по умолчанию, макрос включения пуст {42}. Макрос же отключения вложенных прерываний варьируется в зависимости от параметра **`scmRTOS_SYSTIMER_NEST_INTS_ENABLE`**: {45}{47}.