Системный таймер — SysTick

Мигая светодиодом мы создавали задержку заставляя контроллер обрабатывать цикл. Это никуда не годиться: во-первых, мы не можем контролировать время задержки, подбирать количество операций приходится вручную; во-вторых, микроконтроллер не может делать ничего полезного в это время. Для того, чтобы решить обе эти проблемы, следует использовать таймер.

> Конкретно в предложенной реализации мы решим только первую проблему, подумайте, что нужно сделать, что бы решить вторую.

Подробнее о таймерах в целом, мы поговорим позже, а сейчас рассмотрим самый простой из них, SysTick, содержащийся в ядре Cortex-M микроконтроллера.

Данный таймер является 24-разрядным (т. е. может принимать значения от 0 до 2^{24}-1) и может отсчитывать вниз от заданного значения до нуля, после чего перегружается и генерирует прерывание. Так как таймер содержится в любом Cortex ядре, то его описание вынесено в документацию к ядру — **Cortex-M3 Devices Generic User Guide**, а все необходимые макросы (и даже функции) хранятся в файле `core_<mcu>.h`.

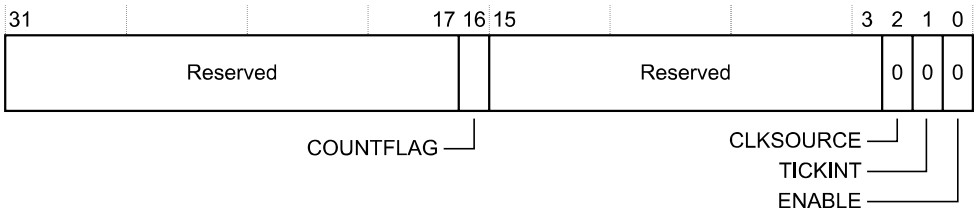
Address	Name	Type	Required privilege	Reset value	Description
0xE000E010	SYST_CSR	RW	Privileged	a	*SysTick Control and Status Register*
0xE000E014	SYST_RVR	RW	Privileged	UNKNOWN	*SysTick Reload Value Register* on page 4-34
0xE000E018	SYST_CVR	RW	Privileged	UNKNOWN	*SysTick Current Value Register* on page 4-35
0xE000E01C	SYST_CALIB	RO	Privileged	-a	*SysTick Calibration Value Register* on page 4-35

a. See the register description for more information.

Таблица со списком регистров системного таймера, Cortex-M3 Devices: Generic User Guide, Table 4-32 System timer registers summary

Регистр контроля и статуса (SYST_CSR)

Как должно быть понятно из названия, данный регистр отвечает за контроль и статус таймера.

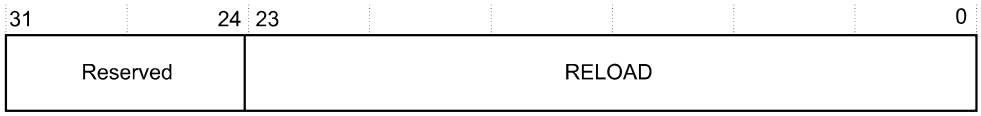


Структура регистра `CSR`

Бит	Название	Функция
[31:17]	–	Зарезервировано
[16]	`COUNTFLAG`	Возвращает `1`, если таймер после последнего считывания перешел `0`
[15:3]	–	Зарезервировано
[2]	`CLKSOURCE`	Указать источник тактирования `0` — внешний, `1` — процессор
[1]	`TICKINT`	Разрешение прерывания `0` — запретить прерывание по достижении `0`, `1` — разрешить прерывание по достижении `0`
[0]	`ENABLE`	Включение или выключение таймера `0` — отключить таймер, `1` — запустить таймер

Регистр значения перезагрузки (SYST_RVR)

Данный регистр отвечает за то, какие значения будут выставлены при перегрузке таймера.

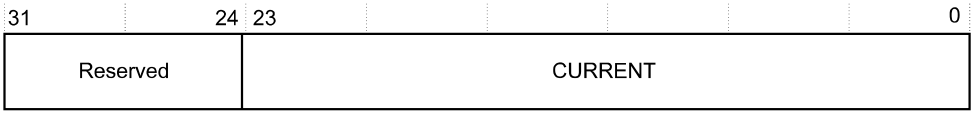


Структура регистра `RVR`

Как видно, из 32 бит используется 24, сюда можно записывать любое число от `0x00000001` до `0x00FFFFFF`. Значение должно быть N-1, т.е. если вы хотите отсчитать 100 тактов, нужно указать 99.

Регистр с текущим состоянием (SYST_CVR)

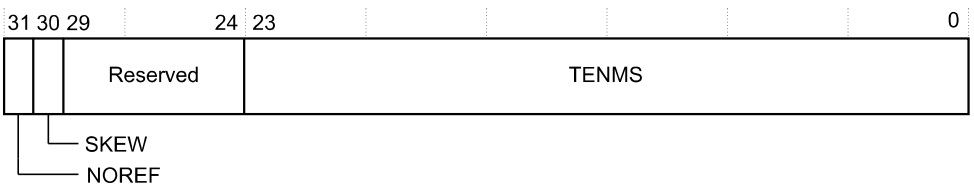
В этом регистре хранится текущее значение таймера.



Структура регистра `CVR`

Регистр калибровки (SYST_CALIB)

Регистр хранит информацию о калибровке таймера.



Структура регистра `CALIB`

Бит	Название	Функция
[31]	`NOREF`	Показывает, есть ли у устройства эталонная тактовая частота `0` — есть `1` — нет Зависит от производителя! Если устройство не поддерживает эталонный источник, то `SYST\_CSR.CLKSOURCE` всегда будет `1` и его нельзя изменить
[30]	`SKEW`	Показывает, является ли `TENMS` точным `0` — точное `1` — неточно или не задано
[29:24]	–	Зарезервировано
[23:0]	`TENMS`	Обновляется каждые 10 мс (100 Гц) и сообщает информацию об ошибке. Если значение `0` — то калибровочное значение неизвестно.

Так как таймер — часть ядра, то он входит в библиотеку CMSIS. Откройте файл `core_<device>.h` и найдите:

```
typedef struct
{
  __IO uint32_t CTRL;
  /*! Offset: 0x000 (R/W) SysTick Control and Status Register */
  __IO uint32_t LOAD;
  /*! Offset: 0x004 (R/W) SysTick Reload Value Register       */
  __IO uint32_t VAL;
  /*! Offset: 0x008 (R/W) SysTick Current Value Register      */
  __IO  uint32_t CALIB;
  /*! Offset: 0x00C (R/ ) SysTick Calibration Register        */
} SysTick_Type;
```

Эта структура таймера. Ниже этой секции будут описаны маски для настройки, например:

```
#define SysTick_CTRL_COUNTFLAG_Pos 16
```

Итак, чтобы настроить таймер, нужно записать в нужные регистры то, что нас интересует. Допустим, частота тактирования нашего микроконтроллера 24 МГц, тогда:

```
SysTick->LOAD = 24000000UL/1000 - 1; // Загрузка значения в 1 ms
SysTick->VAL  = 24000000UL/1000 - 1; // Устанавливаем текущее значение
// Настраиваем таймер
SysTick->CTRL= SysTick_CTRL_CLKSOURCE_Msk |
               SysTick_CTRL_TICKINT_Msk   |
               SysTick_CTRL_ENABLE_Msk;
```

Теперь нужно заглянуть в таблицу прерываний (`startup_<device>.s`):

```
g_pfnVectors:
    .word   _estack
    .word   Reset_Handler
    .word   NMI_Handler
    .word   HardFault_Handler
    .word   MemManage_Handler
# ...
    .word   PendSV_Handler
    .word   SysTick_Handler
# ...
```

Мы знаем, как должна называться функция. Добавим её в `main.c`:

```
void SysTick_Handler(void) {
  // Тут делаем что-то полезное
}
```

В `core_<cpu>.h` имеется функция для настройки таймера — `SysTick_Config`:

```
__STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if ((ticks - 1) > SysTick_LOAD_RELOAD_Msk)  return (1);
  /* Reload value impossible */

  SysTick->LOAD  = ticks - 1;
  /* set reload register */
  NVIC_SetPriority (SysTick_IRQn, (1«__NVIC_PRIO_BITS) - 1);
  /* set Priority for Systick Interrupt */ SysTick->VAL   = 0;
  /* Load the SysTick Counter Value */
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
   SysTick_CTRL_TICKINT_Msk   |
   SysTick_CTRL_ENABLE_Msk;<
  /* Enable SysTick IRQ and SysTick Timer */
   return (0);
  /* Function successful */
}
```

Поэтому настройку таймера можно переписать следующим образом:

```
SysTick_Config(64000000UL/1000);
```

Значение тактовой можно получить из переменной `SystemCoreClock` (объявлена в файле `system_<device>.c`). Перепишем строчку выше следующим образом:

```
SysTick_Config(SystemCoreClock/1000);
```

Добавьте вызов этой функции в `mcu_init()`.

Теперь усовершенствуем нашу программу мигания светодиодом. Нам потребуется переменная под счетчик. Для чего? В нём будет храниться «глобальное время». Поскольку наш таймер генерирует прерывание каждую 1 мс, мы можем записывать в отдельный счетчик количество миллисекунд, которое прошло с момента подачи питания на МК.

```
// utils.c
volatile uint32_t ticks_delay = 0;
// utils.h
extern volatile uint32_t ticks_delay;
```

Спецификатор `volatile` — принуждает компилятор не оптимизировать ваш код, а конкретно заставляет каждый раз считывать значение из адреса где хранится переменная, а не из какой-нибудь временной копии. Нужно это для потому, что значение переменной может быть изменено асинхронно программе, и данные будут просто утеряны. Если `ticks_delay` не будет обновляться, то программа застрянет в функции `delay()`.

Далее в обработчике прерывания будем добавлять «1» к глобальному времени:

```
void SysTick_Handler(void) {
  ticks_delay++;
}
```

Тогда функцию задержки можно реализовать очень просто (добавим её в `utils.c`) — запомнив время вызова функции, можно узнать, сколько времени она выполняется: если разница между текущим временем и временем запуска меньше заданного числа, то ждём, в противном случае выходим из цикла.

```
void delay(const uint32_t milliseconds) {
    uint32_t start = ticks_delay;
    while((ticks_delay - start) < milliseconds);
}
```

Теперь цикл в `main()` можно переписать следующим образом:

```
while(1) {
  led_toggle();
  delay(1000);
}
```

Код урока можно найти на github: **CMSIS**.

Назад | **Оглавление** | **Дальше**