STM8 Microcontrollers. Timers, Part 1.

STM8

STM8 Microcontrollers. Timers, Part 1.

Hello,
Today we will look at timers in STM8 microcontrollers.
Since timers are a broad topic, I decided to split it into a couple of articles for ease of perception and ease of writing. STM8 timers are more sophisticated and complex than AVR timers, but they also provide more capabilities. Let's look at the general characteristics of timers, and along the way we will compare them with their AVR counterparts.
So, STM8 microcontrollers have three types of timers:
- Timer with advanced control (Advanced control) - TIM1;
- General purpose (General purpose) - TIM2, TIM3, TIM5;
- Basic (Basic) - TIM4, TIM6.
The main properties of timers can be found in Table 30 on page **129** of the Reference Manual. If I could lay out the table, I would translate it, but for now we will limit ourselves to a screenshot from the datasheet and comments to it.

Live

Comments Publications

penzet → Sprint Layout in OS X 18 → Software for electronics engineer

Vga → EmBitz 6 → Software for electronics engineer

Vga → Rail-to-rail: ideal operational amplifier or a clever marketing ploy? 1 → Theory, measurements and calculations

Flint → A little more about 1-wire + UART 56 → Hardware connection to the computer.

penzet → Cross-platform terminal - SerIO 3.x 25 → Software for electronics engineer

Gornist → PIP regulator 2 → Algorithms and software solutions

whoim → CC1101, Treatise on Tracing 89 → Blog im. khomin

OlegG → Clock on Bluetooth LE module 8 → Cypress PSoC

Technicum505SU → Nuances of PWM control of a DC motor by a microcontroller 3 → Theory, measurements and calculations

sunjob → DDS synthesizer AD9833 88 → Blog named after grand1987

DIHALT → W801, LCD screen and fly in the ointment 2 → Detail

nictrace → New Arduino-compatible board. 20 → FPGA

sunjob → Connecting the ARM GCC compiler to CLion 1 → Software for electronics engineers

Vga → CRC32: on STM32 as on PC or on PC as on STM32. 58 → STM32

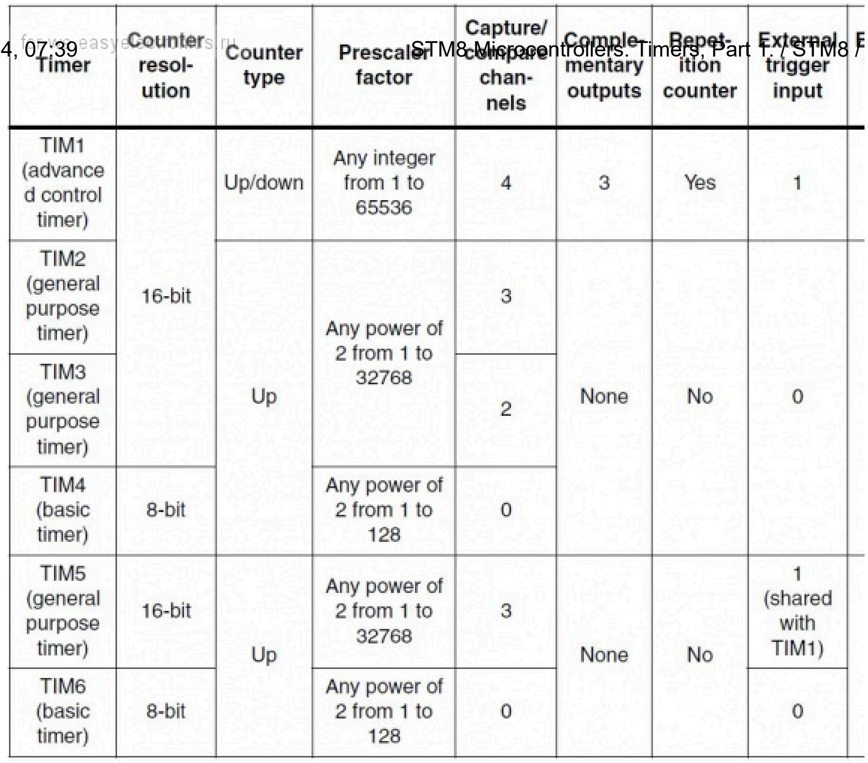
dmitrij999 → Capturing images from a USB camera using STM32 6 → STM32

Gilaks → Expanding the capabilities of a simple MC up to an ADC on 2 or 1 pin. 8 → Theory, measurements and calculations

sva_omsk → Lithium ECAD - Russian PCB CAD 40 → Software for electronics engineer

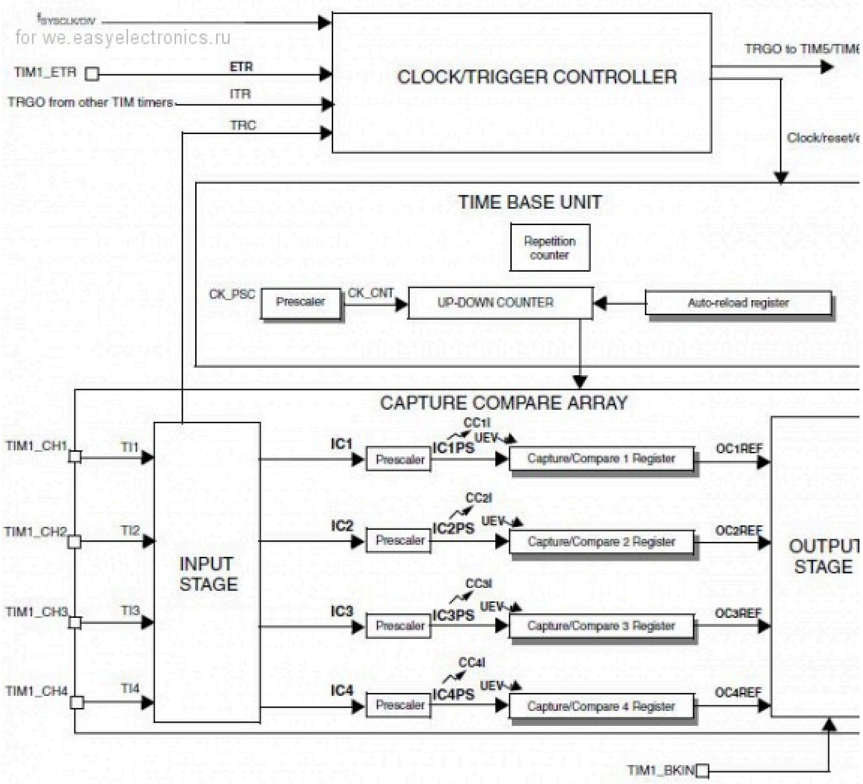
x893 → W801 - budget controller with Wi-Fi 4 → Details

podkassetnik → Changing the standard instrument cluster lighting of Logan-like cars 5 → Automotive electronics



Timer	Counter resolution	Counter type	Prescaler factor	Capture/compare channels	Complementary outputs	Repetition counter	External trigger input	F
TIM1 (advanced control timer)	16-bit	Up/down	Any integer from 1 to 65536	4	3	Yes	1	
TIM2 (general purpose timer)		Up	Any power of 2 from 1 to 32768	3	None	No	0	
TIM3 (general purpose timer)				2				
TIM4 (basic timer)	8-bit		Any power of 2 from 1 to 128	0				
TIM5 (general purpose timer)	16-bit	Up	Any power of 2 from 1 to 32768	3	None	No	1 (shared with TIM1)	
TIM6 (basic timer)	8-bit		Any power of 2 from 1 to 128	0			0	

We will start analyzing the timers with Timer1. It is clear from the table that Timer1 has the largest number of convenient features, which makes it the most convenient, but at the same time the most complex - 80 pages of documentation are devoted to it. Let's look at its block diagram.



Compared to AVR, the ability to select an arbitrary prescaler of the input frequency (Prescaler factor column) and the presence of a stop input immediately catches the eye. Selecting a prescaler not from a series of powers of two allows for flexible configuration of the timer. In addition, the AVR has a maximum prescaler value for all timers - 1024, and for STM8, as we can see from the table - 65535 for Timer1. This allows us to simply measure long time intervals. For example: with a clock rate of 16 MHz and a prescaler of 1024, the time it takes for a 16-bit Timer1 in an AVR to count from 0 to 0xFFFF is (1/16,000,000) * 1024 * 65,535 = 4.19424 s. Using Timer1 in an STM8 and a

Vga → ROPS (Rem Object Pascal Script) - embedded interpreter of the Pascal language. Plugin PSImport_Classes 3 → Algorithms and software solutions

Full broadcast | RSS

1-Wire Altera arduino ARM Assembler Atmel AVR C++ compel DIY enc28j60 ethernet FPGA gcc I2C IAR KEIL LaunchPad LCD led linux LPCXpresso MSP430 nxp PCB PIC pinboard2 RS-485 RTOS STM32 STM8 STM8L TI UART USB algorithm assembler ADC library power unit detail display idea tool contest competition2 LUT microcontrollers for beginners review Debug board soldering iron printed circuit board pay FPGA crafts purchases programmer programming Light-emitting diode software scheme circuit design Technologies smart House photoresist freebie crap Watch humor

Blogs

Top

AVR	**38.98**
STM8	**37.92**
Garbage truck	**29.53**
STM32	**28.46**
Detail	**24.63**
Connection of hardware to the computer.	**24.04**
Circuit design	**18.15**
Smart House	**17.75**
MSP430	**17.13**
LPC1xxx	**14.79**

All blogs

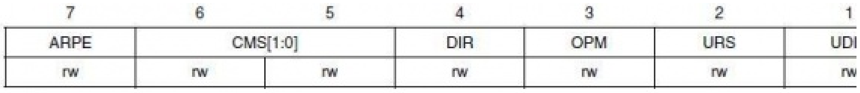
maximum prescaler of 65536, we get: (1/16,000,000) * 65536 * 65,535 = 268.43136 s, which is almost four and a half minutes. An even longer delay can be obtained using a Repetition counter. This is a module that counts timer events. If you configure it to count timer overflows, you get another prescaler with a maximum value of 255. It is also worth noting that the Repetition counter counts down from 255 by default (you can select any value less than 255) to 0. The
stop input can be used in motor control tasks when the timer generates control pulses for the windings, and a temperature sensor or current sensor is connected to the stop input. Then, when the motor is overloaded, it is automatically turned off.

Timer1 has four capture-comparison channels. What are they and how can they be used? In capture mode, these channels allow you to measure the pulse duration at the input for each of the channels. When working in comparison mode, you can generate various delays, as well as form a PWM signal.

Timer1 has a selectable clock source. In addition to the obvious clock from the main frequency of the microcontroller, you can select clock from an external source, as well as clock from other timers. In this case, the timer-clock source acts either as a synchronization source or as an additional prescaler of Timer1.

We will not consider all the configuration registers of Timer1, there are too many of them, we will limit ourselves to the main ones. Information on all the settings and configuration registers of Timer1 can be found in the Reference Manual starting on page **181** .

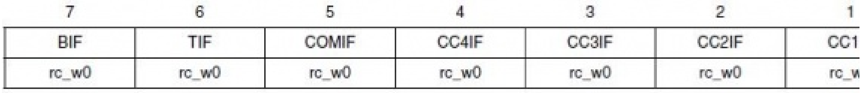
Register **TIM1_CNTR** – *Timer1 Counter* . Timer1 counter register. This is a 16-bit register that stores the current value of the Timer1 counter. This register is accessed via the **TIM1_CNTRH** and **TIM1_CNTRL** registers — *Counter high* and *Counter low* , respectively, with the higher register (**TIM1_CNTRH**) being accessed first (both in read and write mode) , and then the lower register (**TIM1_CNTRL**). The same applies to other 16-bit registers in the STM8S. The **TIM1_PSCR**
register is *the Timer1 Prescaler* . Like **TIM1_CNTR** , this is a 16-bit register, and is accessed in a similar way, via the **TIM1_PSCRH** and **TIM1_PSCRL** registers . The TIM1_PSCR value defines how much the input frequency is divided by according to the formula fCK_CNT = fCK_PSC / (PSCR[15:0]+1), where fCK_CNT is the frequency supplied to the timer, fCK_PSC is the frequency supplied to the divider. The **TIM1_CR1** register is *Control register 1.* Control register 1. It contains the main configuration bits of Timer1. Setting **the CEN** bit starts the timer. Setting **the UDIS** bit disables Timer1 interrupts. **The URS** bit is responsible for selecting the events that will cause an interrupt from Timer1. If this bit is set, the interrupt will occur only when the counter overflows. When the **URS value is 0, the interrupt, in addition to the usual method, can also be caused by the external clock module, or programmatically, by setting the UG** bit in the **TIM1_EGR** register . When **the ORS** bit is set, Timer1 switches to one pulse mode: it counts until overflow and stops. To continue the operation, the timer must be restarted. The **DIR** bit determines the counting directions:



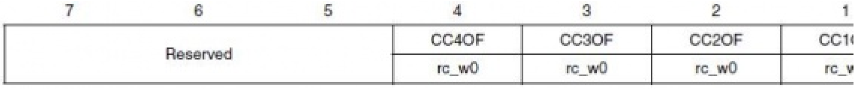
7	6	5	4	3	2	1
ARPE	CMS[1:0]		DIR	OPM	URS	UDI
rw	rw	rw	rw	rw	rw	rw

0 – count up, from 0 to 0xFFFF;
1 – count down, from 0xFFFF to 0. **CMS[1..0]**
bits – select the counting mode; possible values: 00 – the timer counts up or down depending on the state of the **DIR** bit ; 01 – the timer counts up, then down. The comparison register match flag is set when counting down; 10 – the timer counts up, then down. The comparison register match flag is set when counting up; 11 – the timer counts up, then down. The comparison register match flag is set both when counting up and when counting down; **TIM1_CR2** register — *Control register 2* . Control register 2. This register is responsible for configuring the capture and compare module. We will not consider it in detail

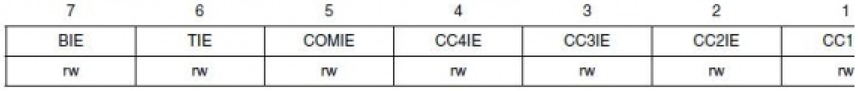
yet. **TIM1_SR1** register — *Status register 1* . Status register 1. The main status register, it contains interrupt flags that must be analyzed when processing interrupts. Let's consider it in more detail: The **UIF** flag becomes equal to one when the counter overflows and the corresponding interrupt is enabled. The **CCxIF** flags are set when the timer counter value matches the value written in the **TIM1_CCRx** register if the timer operates in the compare mode. In the capture mode, these flags are set when a pulse appears on the **ICx** pin , and the current value of the timer counter is written to the **TIM1_CCRx** register . The **BIF** flag is set when a high level appears at the Timer1 stop input. The **TIM1_SR2** register is *Status* register 2. This register is responsible for configuring the capture and compare module. We will not consider it in detail for now. The **TIM1_IER** register is *Interrupt enable register* . **The UIE** bit enables the overflow interrupt. **The CCxIE** bits enable capture and compare interrupts on the corresponding channel. **The BIE** bit enables interruption on the stop input. The **TIM1_EGR** register is the *Event* generation register. An interesting register – setting flags in it leads to manual interrupts that are called automatically as a result of the timer operation. For example, setting the **BG flag** will cause an interrupt on the stop input. The bits of this register are reset automatically, there is no need to clear them manually .



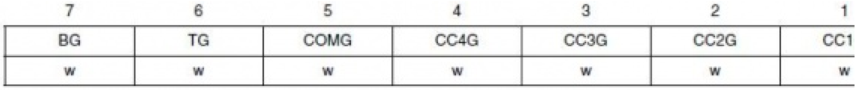
7	6	5	4	3	2	1
BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w



7	6	5	4	3	2	1
Reserved			CC4OF	CC3OF	CC2OF	CC1
			rc_w0	rc_w0	rc_w0	rc_w



7	6	5	4	3	2	1
BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1
rw	rw	rw	rw	rw	rw	rw



7	6	5	4	3	2	1
BG	TG	COMG	CC4G	CC3G	CC2G	CC1
w	w	w	w	w	w	w

We will look at the other registers later.

First, let's set up Timer1 in the simplest and most obvious mode - the up count mode from 0 to 0xFFFF (65535). When the interrupt is triggered, the LED state switches. At the same time, we will study how interrupts are designed. To do this, you just need to zero the configuration registers, enable the overflow interrupt and enable the timer.

So, a little about interrupts in STM8S. Interrupt vectors are located in **UBC** (*User boot code*) and start at address 0x0080000. This memory area is diligently protected from accidental writing, its size is configurable and it can be used, for example, to store the bootloader. Compared to AVR, there are much fewer interrupt vectors, a maximum of two for each peripheral device. This means that when processing an interrupt, it is necessary to analyze which of the possible events caused it.

By default, interrupts are disabled in STM8. Interrupts are enabled by the **RIM** (*Reset Interrupt Mask*) command. Interrupts are disabled by **the SIM** (*Set Interrupt Mask*) command.

Important: the interrupt flags that were set automatically when the interrupt was called are not automatically reset when entering the interrupt handler. This means that when processing interrupts, it is necessary to clear the flag that caused the interrupt, otherwise the interrupt handler will be constantly executed.
In IAR for STM8S, interrupts are formatted as follows:

```
#pragma vector = <номер вектора>
__interrupt void <имя обработчика>(void)
{
        //код обработчика
}
```

We take the vector number from the file iostm8s105s6.h, which we will include in our project. The list of vectors is at the very end of this file. The handler name is chosen arbitrarily, based on the rules of the C language and common sense.

```
#include "iostm8s105s6.h" // подключение заголовочного файла с объявлени
__interrupt void TIM1_OVR_UIF_handler(void);
void init(void)
{
  PD_DDR_bit.DDR0 = 1;   // Ножка PD0 конфигурируется на вывод
  PD_CR1_bit.C10 = 1;   // Выход типа Push-pull
  PD_CR2_bit.C20 = 1;   // Скорость переключения - до 10 МГц.

  //Настройка Таймера1
  // Синхронизация как ведущий с периферией отключена
    TIM1_CR2 = 0;
  // Синхронизация как ведомый с периферией отключена
    TIM1_SMCR = 0;
    // Внешнее тактирование отключено
    TIM1_ETR = 0;
    // Прерывание по обновлению счетного регистра разрешено
    TIM1_IER = MASK_TIM1_IER_UIE;
    // Предделитель - 0
    // ВАЖНО!!!
    // Порядок установки предделителя - старший регистр, потом младший
    TIM1_PSCRH = 0;
    TIM1_PSCRL = 0;

  // Режим непрерывного счета по возрастанию
  // Прерывание по переполнению разрешено и таймер запущен
    TIM1_CR1 = (MASK_TIM1_CR1_URS+MASK_TIM1_CR1_CEN);
}



int main( void ) // Основная программа
{

  init();
  asm("rim"); //Разрешаем прерывания
  while(1)      // Бесконечный цикл
  {
   asm("nop");  //Пустая операция
  }


}

// Вектор прерывания по обновлению или переполнению Таймера1
#pragma vector = TIM1_OVR_UIF_vector
__interrupt void TIM1_OVR_UIF_handler(void)
```

```
{
 // Проверка, что же вызвало прерывание
  if (TIM1_SR1_UIF==1)
  {
    TIM1_SR1_UIF = 0;              // Очистка флага прерывания по обновле
    PD_ODR ^= MASK_PD_ODR_ODR0;    // Переключение уровня напряжения на н
                                   // при помощи операции Исключающее ИЛИ


  }
}
```

The program does not require any special explanation, it is enough to say that the main loop uses the function asm("nop"), which performs an empty operation. It is there for the convenience of debugging. You can conduct several experiments with this program. For example, let's change the value of the prescaler in the lines to some other values:

```
    TIM1_PSCRH = 0;
    TIM1_PSCRL = 100;
```

We see that our LED began to blink less frequently. Now let's see what happens if we violate the order of access to the 16-bit register.

```
    TIM1_PSCRL = 100;
    TIM1_PSCRH = 0;
```

The LED blinks with the frequency it blinked with when the prescaler was zero. Let's open the Watch window (Viev -> Watch) and enter the names of **the TIM1_PSCRL** and **TIM1_PSCRH** registers in the Expression field . We see that, despite the fact that we seem to have written our prescaler value and see their values, in fact, no writing to the register has occurred.
I advise you to conduct another experiment yourself and check what happens if you do not clear the **TIM1_SR1_UIF** flag in the interrupt body.
That's all for today, and next time we will continue to deal with timers.

STM8 , microcontrollers , programming , contest

+3 March 20, 2011, 10:13 PM **Kalvenolt**

Comments (20)

RSS Collapse / Expand

> *It's also worth noting that the Repetition counter counts down from 255 by default (you can choose any value less than 255) to 0.*

Perhaps it would be more correct to say "you can choose any value less than 256"

0

Vishen
March 20, 2011, 10:40 PM

Advanced timer maybe it would sound better as "advanced timer"?

0

Ageofenigma
March 21, 2011, 09:04

But don't you need to add another line like this:

```
CLK_PCKENR1 = 0x80
```

0

?
After all, if my memory serves me right, the timer clock is disabled by default.

SergRuan
01 April 2011, 22:10

Page **67** of the reference manual:
After a device reset, all peripheral clocks are enabled.
In STM32, the peripherals are disabled from clocking by default.

0

Kalvenolt
02 April 2011, 00:34

Exactly. I messed up. I'm just figuring out the STM8L series, and they have everything disabled by default. Probably to save energy.

0

SergRuan
02 April 2011, 09:33

To do the same task for STM8L series, you just need to add CLK_PCKENR2 = 0x20 or something else?

0

Onion
11 February 2012, 22:43

I have a question, are the counters asynchronous there?
Because in AVR it's a total pain with these synchronous counters...
you can't supply more than a quarter of the core clock.
And you can't make a simple frequency meter.

0

selevo
18 February 2012, 12:03

The entire AVR line was ruined by this crap, by putting a trigger at the counter input.
There is a clock one, but I don't think it will handle it.

0

selevo
18 February 2012, 12:06

~~With the ORS~~ bit set , maybe **OPM** ?

0

DOOMSDAY
May 28, 2012, 12:53 PM

And should the prescaler be reset before starting the timer, as in AVRs, or not
???

0

ingor-ru
09 August 2012, 19:10

I don't get it, I'm torturing timer 1 with an oscilloscope, and somehow it turns out that the timer doesn't pay any attention to the prescaler. That it equals 0, that it equals 0xffff. Why can this be?

0

urdmitriy
04 October 2012, 11:08

Well, firstly, the table talks about values from 1 to 65535. It is not a fact that it will accept a zero. And secondly, do you follow the order of writing to a 16-bit register? At the very end of the article there is an example very similar to the behavior you describe.

0

Vga
04 October 2012, 13:51

Yes, that's all clear. No, I wrote absolutely different values, with the senior byte first, the values are written to the register (according to the debugger). In general, something is happening with my controller, I need to try to transfer it. The stack started to break out of nowhere.

0

urdmitriy
05 October 2012, 08:29

I spent almost the whole day trying to figure out timer 4, but for some reason it didn't want to start, so I decided to try timer 1 with the same settings. In an infinite loop, I copy the value of TIM1_CNTR to a variable that is visible in the debugger, the variable is constantly increasing, which means the timer is ticking, I try to start timer4:

0

```
TIM4_CNTR = 0;
        TIM4_PSCR = 1;
        TIM4_CR1 = MASK_TIM4_CR1_CEN;


        while(1)              // Бесконечный цикл
        {
                gh = TIM4_CNTR;
        }
```

the variable gh always remains equal to 0, sometimes, for some reason, it is equal to 1. What is the problem?

MrMisha
01 November 2012, 17:08

Hi! Great articles about MK! But I have a question: how should I search for an application note (AN) on the ST.COM website?? I can't do it at all - there are a whole bunch of documents there, but not what I need. What should I type in the search bar to find AN for timers, GPIO, etc. Please share your experience)

0

Juli05
April 27, 2015, 14:43

Here on the left in the Resources section you can find all the available information. AN on ports and timers - are you serious?) Such things are taken from the datasheet (in the case of ST, this is the reference manual). If we are talking about SPL, then it comes with a chm file where all the functions are chewed up, and all the sources are open for study.

0

1essor1
April 27, 2015, 20:24 ↑

Wow, thanks. For some reason I didn't look at the reference manual, I looked at the datasheet. Found what I wanted.)

0

Juli05
April 28, 2015, 07:35 ↑

Only registered and authorized users can leave comments.

Design by — Studio XeoArt

© Powered by LiveStreet CMS