

All Collective Personal TOP

Good Bad

Поиск

8L-Course, Part 5 - Timers: The Beginning

STM8

HIGH-QUALITY PCB

Free shipping

ONLY \$5

FOR 10 PIECES

- Rogers, HDI, aluminum and rigid-flex PCB are available now
- Production time 24 hours

PCB ASS

Free shipping

ONLY

- Component
- Quality a

[← Part 4 — Clocking Contents](#) [Part 6 — Timers, External Channels](#) →

Timers are one of the most important elements of the MCU. No more or less complex program can do without them. Everything that is somehow connected with time intervals or counting any events is implemented using timers.

The STM8L has several timers of varying complexity. They can be divided into three groups:

TIM1 — This is the most complex and functional timer. **16-bit** counter (maximum value — 65535), **prescaler accepting any values from 1 to 65536**. Three external channels with complementary outputs for each... and other goodies.

TIM2, TIM3, TIM5 (the last one is not in our MCU) — A little simpler. There are only two external channels. The divider no longer accepts any value, but only powers of two in the range from 1 to 128. The counter is still 16-bit.

TIM4 is the simplest. 8-bit counter, divider - (also powers of two) from 1 to 32768. There are no external channels.

Here is a summary table from the reference manual, which tells about the characteristics of all timers:

18.1 Timer feature comparison

for we.easyelectronics.ru

Table 65. Timer feature comparison

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

Let's start with TIM1 . It is certainly more complicated than all the others, but

Live

Comments

Publications

[penzet](#) → [Sprint Layout in OS X 1.8](#) → [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](#)

[From Gorn](#) → [PIP regulator 2](#) → [Algorithms and software solutions](#)

[whom](#) → [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 tar spoon 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](#)

[Gilak](#) → [Expanding the capabilities of a simple MC up to an ADC on 2 or 1 pin. 8](#) → [Theory, measurements and calculations](#)

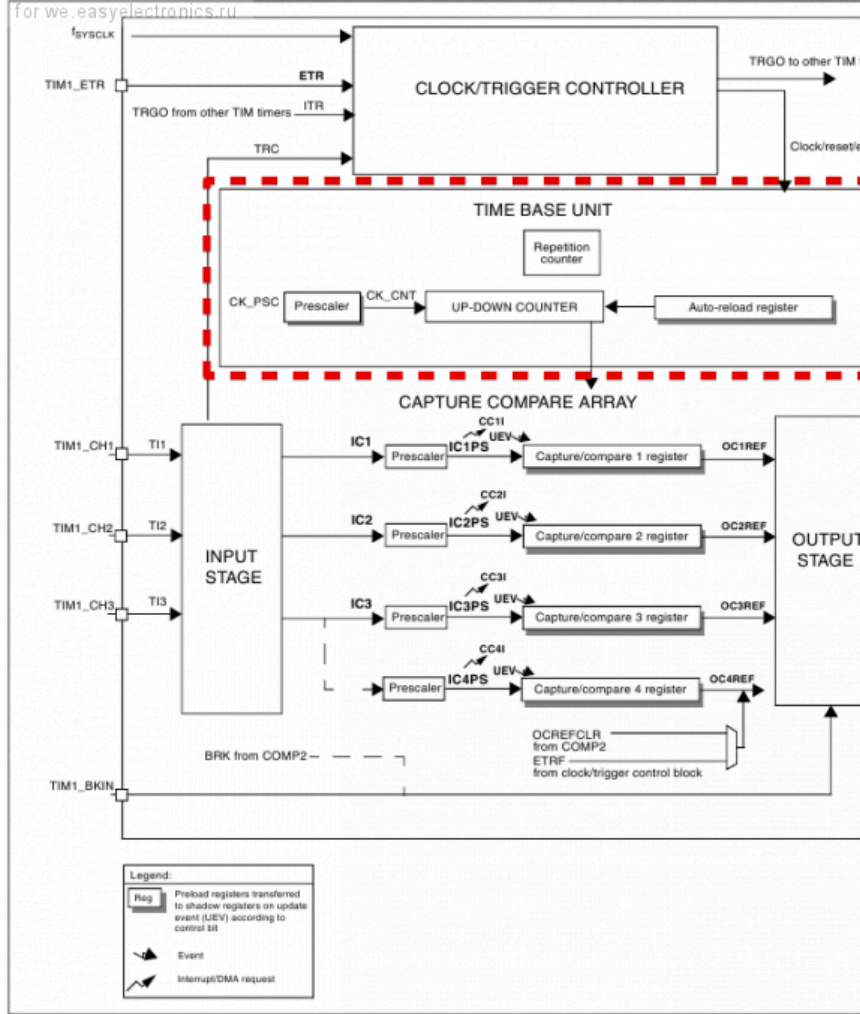
[sva_omsk](#) → [Lithium ECAD - Russian PCB CAD 40](#) → [Software for electronics engineer](#)

[x893](#) → [W801 - budget controller with Wi-Fi 4](#) → [Details](#)

[podkassetnik](#) → [Changing the standard instrument cluster lighting of Logan-like cars 5](#) → [Automotive electronics](#)

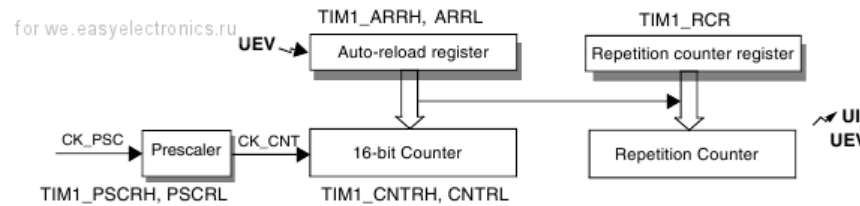
In order not to overload the reader's brain too much, in this part we will consider only the basic capabilities of the timer. The functional diagram of TIM1 brings horror and confusion to the head of an unprepared viewer, so I have highlighted the parts that we will get acquainted with today:

Figure 67. TIM1 general block diagram



A little larger and without JPEG compression

Today we are interested only in **TIME BASE UNIT** . It contains a prescaler and a counter.



The timer receives a clock signal. If we did not touch the CLOCK/TRIGGER CONTROLLER settings, then this will be **the system clock signal SYSCLK** (from which the core and most peripherals work).

As with any other peripheral device, the timer **must be clocked before it starts working** . The **PCKEN21** bit in the PCKENR2 register is responsible for this.

```
PCKENR2 |= PCKEN21; //Тактирование подано
```

Then the clock signal goes through the prescaler. **For TIM1, the prescaler can take any value from 1 to 65536** (and not just powers of two. In general, it is

1-Wire The other arduino ARM
Assembler Atmel AVR C++ compel
DIY enc28j60 ethernet FPGA gcc I2C
AND KEIL LaunchPad LCD led linux
LPCXpresso MSP430 npx PCB PIC
pinboard2 RS-485 RTOS STM32
STM8 STM8L OF UART USB
algorithm assembler ADC the library
power unit detail display an idea tool
competition competition 2 ANGRY
microcontrollers for beginners review
Debug board soldering iron
printed circuit board salary FPGA crafts
purchases programmer programming
Light-emitting diode software scheme
circuit design Technologies
smart House photoresist it's free crap
Times humor

Blogs

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](#)

The prescaler is stored in the TIM1_PSCRH and TIM1_PSCRL registers .

The first one contains the most significant byte, the second one contains the least significant byte. The real value of the prescaler is one more than what is written in the registers. That is, if we write 0 to TIM1_PSCRH:TIM1_PSCRL, the prescaler will be 1. If 65535, the prescaler will become = 65536. When writing to the prescaler, you must first write the most significant byte (TIM1_PSCRH), and then the least significant byte.

After the divider, the signal goes to the counter, which, depending on the settings, counts up (increasing with each clock pulse) or down. The DIR bit in the TIM1_CR1 register is responsible for the direction of the counter. If it is set, the timer counts down, and if it is cleared (by default), then it counts up.

The counter value can be read at any time from the TIM1_CNTRH and TIM1_CNTRL register pair. You must first read the high byte (CNTRH, while the low byte is loaded into a temporary buffer), and then the low byte. You can also change the counter value at any time, and it makes no difference which byte is written first: there is no buffering when writing. Therefore, you should think twice before changing the counter value when the timer is running - otherwise, there is a chance of accidentally catching an interrupt triggering between writing one register and another.

When counting **up** , the timer ticks to the value in the Auto Reload Register (ARR). When the counter matches the ARR value, a reset occurs, after which the timer continues counting starting from 0.

And when counting **down** , it counts to 0, and then resets and starts counting from the value in ARR.

In some cases, the timer needs to stop after the first overflow (and be started only manually). There is an OPM bit in the same TIM1_CR1 for this. If you write 1 to it, the timer will be disabled after the first overflow.

ARR, like the counter, occupies two registers: TIM1_ARRH, TIM1_ARRL. But unlike the counter, there is no buffer for reading (registers can be read in any order), but there is buffering when writing - first you need to write the high byte, and then the low byte.

When the counter value matches the ARR value, in addition to resetting the counter, an Update Event (UEV) is generated. This event can cause an interrupt, the vector of which is called TIM1_OVF_UIF_vector

```
ISR(TIM1_OVF, TIM1_OVF_UIF_vector)
{
    //Заготовка для обработчика прерывания

    TIM1_SR1_bit.UIF = 0; //Сброс флага прерывания
};
```

To enable this interrupt, you need to **set the UIE bit in the TIM1_IER register.**

And also, according to UEV, a new prescaler value is loaded into the timer. That is, if we changed them before, the new values will take effect only at the next UEV.

Even during timer initialization, when we configure the prescaler, we need UEV so that it is written to the timer. Fortunately, we are given the ability to generate timer events programmatically. In the TIM1_EGR register, there is a UG bit, by setting which we simulate the event we need. And to prevent an interrupt from occurring at this moment (which is clearly out of place during timer initialization), we set the URS bit in TIM1_CR1 - if it is equal to 1, then only a counter overflow (or rather a coincidence with ARR) will lead to an interrupt, and software generation of UEV will not.

The CEN bit in TIM1_CR1 is used to start the timer. By setting it, we start the timer (and by resetting it, we stop the counting).

The frequency with which the timer outputs UEV (unless, of course, the OPM bit is set and it is a constant timer) is equal to $\text{SYSCLK} / \text{PRESCALLER} / \text{ARR}$. For example, if we want to receive interrupts 10 times per second with a clock rate of 16 MHz, then the prescaler should be set to 160 (and write the number 159 to the registers for this!), and ARR — 10000. Or vice versa :)

For example, the timer initialization can be like this:

```
CLK_PCKENR2_bit.PCKEN21 = 1; //Включаем тактирование таймера 1
TIM1_PSCRH = 0;
TIM1_PSCRL = 159; //Делитель на 160
TIM1_ARRH = (10000) >> 8; //Частота переполнений = 16M / 160 / 10000 =
TIM1_ARRL = (10000)& 0xFF;

TIM1_CR1_bit.URS = 1; //Прерывание только по переполнению счетчика
TIM1_EGR_bit.UG = 1; //Вызываем Update Event

TIM1_IER_bit.UIE = 1; //Разрешаем прерывание
TIM1_CR1_bit.CEN = 1; //Запускаем таймер
```

There is just one important point - this initialization is designed for the timer not to be touched before it. That is, all registers have default values, remaining from the moment of reset. And if the timer is reconfigured from another mode - you need to make sure that the "extra" settings are reset.

For example... let's blink the LED again :) This time through an interrupt from the timer.

We will do the initialization as shown above, but change the prescaler from 160 to 1600 - then the interrupt will occur once a second:

```
TIM1_PSCRH = (1599) >> 8;
TIM1_PSCRL = (1599)& 0xFF;
```

In the timer interrupt we will blink the LED:

```
ISR(TIM1_OVF, TIM1_OVR_UIF_vector)
{
    PD_ODR_bit.ODR4 ^= 1; //Инвертируем пин со светодиодом (это для модуля

    TIM1_SR1_bit.UIF = 0;
};
```

That's it. If you don't go into all the timer tricks right away, there's nothing complicated.

All timers are built roughly the same way, so you can already work with TIM2, TIM3 (their prescaler is calculated differently!) or TIM4. Just keep in mind that for them, clocking is enabled by other bits:

9.14.4 Peripheral clock gating register 1 (CLK_PCKENR1)

Address offset: 0x3

Reset value: 0x00

7	6	5	4	3	2	
PCKEN1[7:0]						
rw	rw	rw	rw	rw	rw	r

Bits 7:0 **PCKEN1[7:0]**: Peripheral clock enable

These bits are written by software to enable or disable the SYSCLK clock to the peripheral. See [Table 19](#)

0: SYSCLK to peripheral disabled

1: SYSCLK to peripheral enabled

Table 19. Peripheral clock gating bits (PCKEN 10 to PCKEN 17)

Control bit	Peripheral
PCKEN17	DAC
PCKEN16	BEEP
PCKEN15	USART1
PCKEN14	SPI1
PCKEN13	I2C1
PCKEN12	TIM4
PCKEN11	TIM3
PCKEN10	TIM2

That's all for now. In the next part, we'll study working with external timer channels.

Project in IAR

← [Part 4 — Clocking Contents](#) Part 6 — Timers, external channels →

STM8 , STM8L, Timers

+13

March 13, 2013, 10:07 p.m

dcoder

1

Files in the topic: [5_TIM.zip](#)

Comments (23)

[RSS](#) [Collapse](#) / [Expand](#)

It seems to be this:

When writing to the prescaler, you must first write the high byte (TIM1_PSCRH), and then the low byte.

Somehow it doesn't fit with this:

We will do the initialization as shown above, but we will change the prescaler from 160 to 800 - then the interrupt will occur twice a second:

```
TIM1_PSCRL = (1600)& 0xFF;  
TIM1_PSCRH = (1600) >> 8;
```

Well, since we're talking about TIME BASE UNIT, it would probably be worth mentioning the repetition counter, so as not to come back to it later.

But anyway - thanks!



Victor
14 March 2013, 03:45

Thank you, fixed

0

TIM4, TIM5 (the last one is not in our MK) - The simplest.


0

Hmm, and the plate claims that TIM5 is a GP timer, not a basic one.

TIM1_PSCRL = 160; //Делитель на 160

TIM1_PSCRH = 0;

The order has already been mentioned, but I will point out that you set the divider to 161, not 160. Well, in the second case, the prescaler is set not to 800, but to 1601.

 Vga

14 March 2013, 04:00

>_<

0

I wish I would hurry to post an article again... never

 dcoder

March 14, 2013, 3:21 p.m

↑

TIM1_PSCRH = (1559) >> 8;

TIM1_PSCRL = (1559) & 0xFF;

0

You're still glitching)

 Vga

March 14, 2013, 3:45 p.m

↑

And there's also

0

TIM1_ARRH = (10000) >> 8;

A small thing, of course...


 angel5a

March 14, 2013, 5:33 p.m

↑

Hm, what's the problem? (10000) is converted to 8 bits and the result is nonsense?

0

 Vga

March 14, 2013, 6:47 p.m

↑

Exactly the same as at the beginning of the thread. The counting is in the range [0;ARR], i.e. with a coefficient of 10000 there will be a division by 10001.

0

That's why in the comment below I referred to the illustrations, they are more obvious.

The error is 2 orders of magnitude smaller, but suddenly someone decides to make a frequency meter :)

 angel5a

March 15, 2013, 10:59 am

↑

Oh well, the article is good. Well, there are some minor flaws, but we'll fix them.

0

And we're already starved for your articles, we can't wait for the next one to come out :)

 angel5a


March 15, 2013, 11:01 am

↑

+1

0

I can't wait for my STM8S003 with ST-LINK to arrive)

 khomin

14 March 2013, 06:54

When counting up, when the counter matches the ARR value, a reset occurs, after which the timer continues counting starting from 0 (and when counting down, from 65535).

For STM8S/A RM0016 17.3.5. states that when counting down, the counting goes from ARR to 0. Is this different behavior of the S and L series, or a typo?

Regarding double buffering of registers, their illustrations in the datasheet (in the specified reference 38-39) are of great help. I understand that for one article this is already too much, but in the future it would be necessary to indicate that it is possible without buffering.



angel5a

March 14, 2013, 10:38 am

And when counting up from 0 to ARR or from ARR to overflow? Logically, the register name should be the second, but who knows, ST engineers. They are a bit strange.

0



Vga

March 14, 2013, 1:06 p.m



from 0 to arr. The name can be adjusted to the answer that from arr the upper limit of the account will be automatically overloaded :)

0



angel5a

March 14, 2013, 5:35 p.m



There is such a problem. Let's say you need to make a time delay using a 16-bit timer, but not using interrupts from it. For STM32, the code looks like this:

```
[операторы]
while(TIM3->CNT<500){};
[операторы]
```

In the case of stm8, is it necessary to compare the senior and junior bytes of the timer's counter register in turn? I remember that in AVR's there was an option to directly access two 8-bit registers as one 16-bit one. If anyone has a piece of such code for stm8, please share it.

0



NBS

04 September 2013, 01:08

Answering my own question:

```
while ((TIM3->SR1 & TIM3_SR1_UIF) == 0){};
```

Naturally, you should first write the required values into the ARRH and ARRL registers and set up the prescaler.

0

NBS

07 September 2013, 16:44



Guys, tell me why the UG flag is not set (on Discovery)?

The timer initialization is copied exactly from the article.

Of course, I understand that UG is UG, but still...

0



JokerDVB

October 20, 2013, 00:03

for we.easyelectronics.ru

```
TIM1_ARRL = LO(autoreload);
```

```
TIM1_CR1_bit.URS = 1; // Прерывание только по переполнению счетчика
TIM1_EGR_bit.UG = 1; // Вызываем Update Event, чтобы настройки таймера
```

```
TIM1_IER_bit.UIE = 1; // Разрешаем прерывание по Update Event
TIM1_CR1_bit.CEN = 1; // Запускаем таймер
asm("rim");
```

0

Have you tried looking into the refman? It is clearly written there:
1) The UG bit is reset by hardware
2) The UG bit is only available for writing
Also, this could have been guessed without the refman.

0

Vga

20 October 2013, 01:38

SPs)

0

JokerDVB

October 20, 2013, 14:00

Thank you, dcoder, you made my coursework :)
More precisely, I made it after reading all these articles and examples - and a couple of months ago I didn't even know which side to approach microcontrollers from)

0

Kuznets

January 29, 2014, 20:57

In about 2 years the description will reach USART

0

x893

April 15, 2014, 14:46

You are an optimist, however.

0

Vga

April 15, 2014, 19:55

It's a pity there is no continuation anywhere, capture on timers.

0

zelya

June 14, 2015, 1:25 PM

Only registered and authorized users can leave comments.