

All Collective Personal TOP

Good Bad

Поиск

## STM8 microcontrollers. Clocking system.

STM8



## STM8 microcontrollers. Clocking system.

Hello,

Last time we started to consider timers, and today we will figure out how the clocking system in STM8S is arranged.

Compared to AVR, STM8 wins greatly in power and flexibility of clocking. The only disadvantage is the inability to clock from quartz with a frequency of less than 1 MHz, but this is compensated by the presence of an internal low-frequency generator. The main advantage of STM8 over AVR is the absence of FUSE bits! All clocking parameters are configured directly during the program operation.

So, our clock frequency sources can be:

- an external quartz resonator with a frequency of 1-24 MHz ( **HSE** - *High Speed External* );
- an external frequency source of 1-24 MHz ( **HSE-ext** - *High Speed External* );
- an internal high-speed RC generator 16 MHz ( **HSI** - *High Speed Internal* );
- an internal low-speed RC generator 128 kHz ( **LSI** - *Low Speed Internal* ).

Here is the block diagram of the clock module:

### 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](#)

**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 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](#)

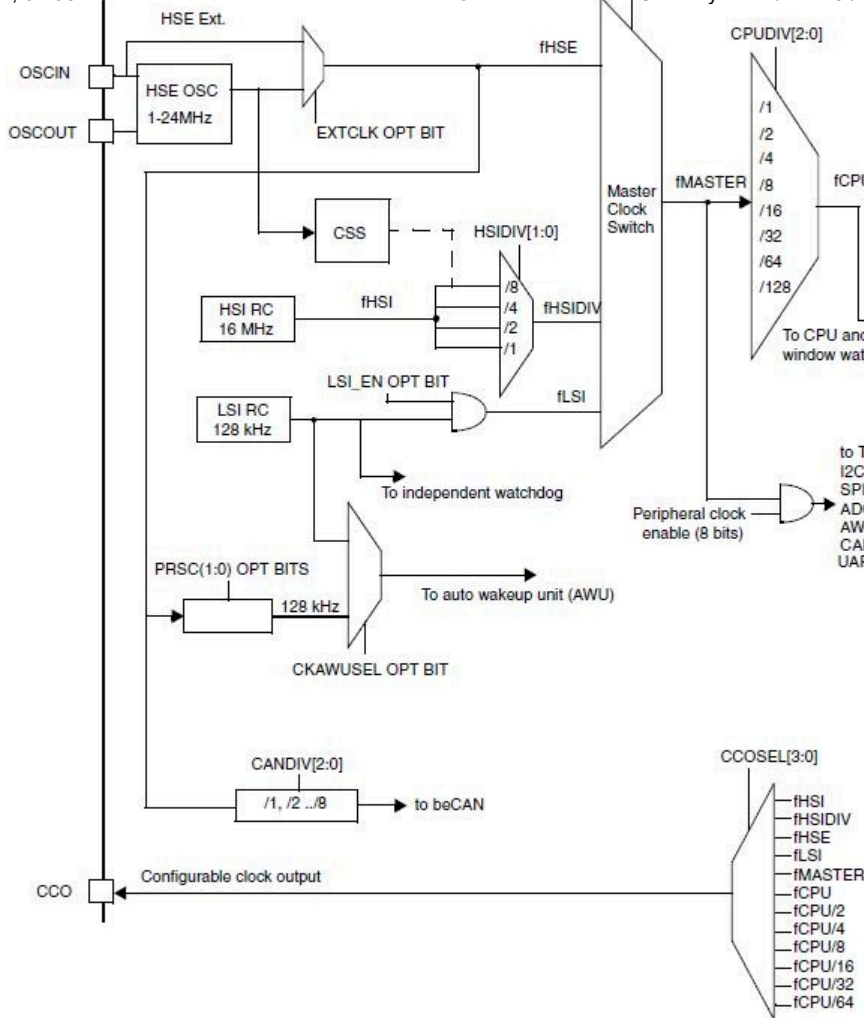
**dmitrij999** → [Capturing images from a USB camera using STM32 6](#) → [STM32](#)

**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](#)



In the diagram, we see that the main frequency is selected in the **Master Clock Switch** module. After that, it goes to the peripheral devices (timers, SPI, etc.) and through the divider to the CPU. When turned on, the main source of the clock frequency is the HSI with a divider of 8, which gives us an operating frequency after power is applied of  $16/8 = 2$  MHz. It is logical that when turned on, the processor starts working from the internal generator - after all, there may not be an external one. The frequency is reduced so that the processor can start up in a bad power supply state.

The low-speed generator can both clock the CPU and work independently of the main clock sources, controlling the independent watchdog timer (**IWD**) and the module for exiting the sleep mode.

The clock system module can cause two interrupts - when switching the clock source (*Master clock source switch event*), and when the clock security system is triggered (*Clock Security System event*).

The signal from any clock source, regardless of whether it is currently active or not, can be routed outside the controller to the **CCO** pin.

The clock source can be switched manually or automatically. The switching methods can be found in the Reference Manual on page **64**. I switch as follows:

1. Write the value corresponding to the required clock source to the **CLK\_SWR register**.
2. Wait for the frequency source to stabilize by monitoring the **SWIF** bit in the **CLK\_SWCR** register.
3. Enable source switching by setting the **SWEN** bit in the **CLK\_SWCR** register.
4. Write the value corresponding to the required clock frequency source to the **CLK\_SWR register**. In this case, the **SWBSY** bit in the **CLK\_SWCR** register is set and the corresponding clock source is started. At this point, the microcontroller is still clocked from the old source. After the frequency stabilizes, the value from the **CLK\_SWR** register is copied to **CLK\_CMSR** and

1-Wire Altera arduino ARM Assembler  
 Atmel AVR C++ compel DIY enc28j60  
 ethernet FPGA gcc I2C IAR KEIL  
 LaunchPad LCD led linux LPCxpresso  
 MSP430 npx 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](#)

|   |              |
|---|--------------|
| <a href="#">AVR</a>                                     | <b>38.98</b> |
| <a href="#">STM8</a>                                    | <b>37.92</b> |
| <a href="#">Garbage truck</a> 🗑️                        | <b>29.53</b> |
| <a href="#">STM32</a>                                   | <b>28.46</b> |
| <a href="#">Detail</a>                                  | <b>24.63</b> |
| <a href="#">Connection of hardware to the computer.</a> | <b>24.04</b> |
| <a href="#">Circuit design</a>                          | <b>18.15</b> |
| <a href="#">Smart House</a>                             | <b>17.75</b> |
| <a href="#">MSP430</a>                                  | <b>17.13</b> |
| <a href="#">LPC1xxx</a>                                 | <b>14.79</b> |

[All blogs](#)

Let's consider the control registers of the clock module: **CLK\_CMSR** register - *Clock master status register* . The register of the main frequency status. This register stores information about which clock source is currently current. The register is read-only, writing to this register will not produce any result. Possible values of the register:

- 0xE1: Current clock source – HSI (value after reset);
- 0xD2: Clock source – LSI;
- 0xB4: Clock source – HSE.

Register **CLK\_SWR** — *Clock master switch register* . Register of the main frequency switching. It is written to in order to change the clock frequency source. Constants for its selection are the same as in the **CLK\_CMSR** register . Writing to this register is impossible during the frequency switching process (at this moment **the SWBSY** bit in the **CLK\_SWCR** register is set ).

Register **CLK\_CKDIVR** — *Clock divider register* . Frequency divider register.

| 7        | 6 | 5 | 4           | 3  | 2      | 1  |
|----------|---|---|-------------|----|--------|----|
| Reserved |   |   | HSIDIV[1:0] |    | CPUDIV |    |
|          |   |   | rw          | rw | rw     | rw |

Bits **HSIDIV[1:0]** select the divider for the internal generator. Possible values:

- 00 – the divisor is 1 (the frequency is not divided);
- 01 – the divisor is 2;
- 10 – the divisor is 4;
- 11 – the divisor is 8.

**The CPUDIV[2:0]** bits define the divisor for the CPU.

Possible values are:

- 000 — fCPU=fMASTER
- 001 — fCPU=fMASTER/2
- 010 — fCPU=fMASTER/4
- 011 — fCPU=fMASTER/8
- 100 — fCPU=fMASTER/16
- 101 — fCPU=fMASTER/32
- 110 — fCPU=fMASTER/64
- 111 — fCPU=fMASTER/128

**The CLK\_ICKR** register is *an Internal clock register* . The internal clock source control register. This register is responsible for the HSI and LSI generator settings. Let's look at its bits: The **HSIEN**

| 7        | 6 | 5     | 4      | 3     | 2   | 1      |
|----------|---|-------|--------|-------|-----|--------|
| Reserved |   | REGAH | LSIRDY | LSIEN | FHW | HSIRDY |
|          |   | rw    | r      | rw    | rw  | r      |

bit is responsible for enabling the high-frequency generator. By default, it is equal to one and the generator is enabled. The **HSIRDY** bit determines the state of the HSI generator. If it is set, the generator is ready for operation. Setting **the FHWU** bit enables fast exit from sleep modes. The functions of **the LSIEN** and **LSIRDY** bits correspond to those for the HSI and are responsible for the operation of the low-speed generator. **The CLK\_ECKR** register is *an External clock register*. The external clock source control register. This register is responsible for the HSE generator settings. It has only two configuration bits:

**HSEEN** - setting this bit enables the external clock source (quartz or generator); **HSERDY** - this bit is set by hardware and indicates the readiness of the external source for operation. The **CLK\_SWCR** register is *a Switch control register* . Clock frequency switching control register. Let's consider its constituent bits: **SWBSY** – this bit is set by hardware at the moment of the clock frequency source switching process. **SWEN** – setting this bit enables clock frequency source switching. Setting the **SWIEN** bits enables the clock frequency source switching interrupt. The **SWIF** flag is set after the source switching. It must be cleared in the interrupt handler procedure. **CLK\_CSSR** register — *Clock security system register* . Clock security system control register. **The CSSEN** bit enables the **CSS** module . **The CSSDIE** bit enables the interrupt call for the

|          |   |   |   |       |       |     |
|----------|---|---|---|-------|-------|-----|
| 7        | 6 | 5 | 4 | 3     | 2     | 1   |
| Reserved |   |   |   | SWIF  | SWIEN | SWE |
|          |   |   |   | rc_w0 | rw    | rw  |

|          |   |   |   |       |        |     |
|----------|---|---|---|-------|--------|-----|
| 7        | 6 | 5 | 4 | 3     | 2      | 1   |
| Reserved |   |   |   | CSSD  | CSSDIE | AUX |
|          |   |   |   | rc_w0 | rw     | r   |

It's time to practice on cats. The STM8S - Discovery board has a 16 MHz quartz, so we'll use it. Let's change the program from the previous article about timers: let's add switching of the clock to the quartz. At the same time, let's increase the prescaler of Timer1 to see the LED switching, since the timer clock frequency will increase from 2 MHz (default value) to 16 MHz, and the LED blinking will be invisible to the eye.

```
include "iostm8s105s6.h" // подключение заголовочного файла с объявления
__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;
    // ВАЖНО!!!
    // Порядок установки предделителя - старший регистр, потом младший
    TIM1_PSCRH = 0;
    TIM1_PSCRL = 10;

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

    // Настройка тактового генератора
```

```

CLK_ECKR_bit.HSEEN = 1;           // Включаем HSE
CLK_SWCR_bit.SWEN=1;             // Включаем внутренний делитель частоты
while(CLK_ECKR_bit.HSERDY != 1) {} // Ждем готовности источника такти
CLK_CKDIVR = 0;                   // Предделитель равен нулю
CLK_SWR = 0xB4;                   // Выбираем HSE источником тактов
while (CLK_SWCR_bit.SWIF != 1){}  // Ждем готовности переключения
}

int main( void ) // Основная программа
{
    init();
    asm("rim");
    while(1) // Бесконечный цикл
    {
        asm("nop");
    }
}

// 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
    }
}

```

STM8 has such a wonderful module as **CSS** ( *Clock Security System* ). It monitors the operation of the quartz generator, and can, in case of its failure, switch the clock to the internal generator. In this case, the input frequency divider is set to 8, which corresponds to 2 MHz. Also, at the moment of switching, the **CSSD** ( *Clock security system detection* ) flag is set in the **CLK\_CSSR** register and an interrupt is called for the operation of the clock security system ( *Clock security system detection interrupt* ), if it is enabled by the **CSSDIE** ( *Clock security system detection interrupt enable* ) flag. This module is enabled by setting the **CSEEN** ( *Clock security system enable* ) bit in the **CSSR** register . Let's edit our program so that in case of quartz failure, it automatically switches to **HSI** , calls an interrupt and performs some actions.

And here I encountered an error in the programming environment. Let me remind you that I am using IAR Embedded Workbench for STMicroelectronics STM8, version 1.20. In the iostm8s105s6.h header file that I use, there is the following description of the **CLK\_CSSR** register :

```

/* Clock security system register */
#ifdef __IAR_SYSTEMS_ICC__
typedef struct
{
    unsigned char CSEEN      : 1;
    unsigned char AUX        : 1;
    unsigned char CSSD       : 1;
    unsigned char CSSDIE     : 1;
} __BITS_CLK_CSSR;

```

We look at the description of this register in the Reference manual on page **77** and are surprised: the **CSSD** and **CSSDIE** bits are swapped! Let's correct the contents of this file.

```
/* Clock security system register */
#ifdef __IAR_SYSTEMS_ICC__
typedef struct
{
    unsigned char CSSEN      : 1;
    unsigned char AUX        : 1;
    unsigned char CSSDIE     : 1;
    unsigned char CSSD       : 1;
} __BITS_CLK_CSSR;
#endif
__IO_REG8_BIT(CLK_CSSR, 0x50C8, __READ_WRITE, __BITS_CLK_CSSR);
```

And we'll fix the masks a little lower.

```
#define MASK_CLK_CSSR_CSSDIE    0x04
#define MASK_CLK_CSSR_CSSD     0x08
```

In general, this error is also present for the entire iostm8.h header family. I haven't looked at the headers for other processors, but I suspect that the same error is present there. In addition, iostm8s105s6.h lacks interrupt vector numbers for the clock system. Let's take them from iostm8.h and insert them at the end of our header. They look like this:

```
#define AWU_vector              0x03
#define CLK_CSS_vector         0x04
#define CLK_SWITCH_vector      0x04
```

And with the already corrected header file, we will write our program.

```
#include "iostm8s105s6.h" // подключение заголовочного файла с объявлени.
__interrupt void TIM1_OVR_UIF_handler(void);
__interrupt void CLK_CSS_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 = 10;
```

```

TIM1_CR1 = (MASK_TIM1_CR1_URS+MASK_TIM1_CR1_CEN);

// Настройка тактового генератора
CLK_ECKR_bit.HSEEN = 1;           // Включаем HSE
CLK_SWCR_bit.SWEN=1;             // Разрешаем переключение источни
while(CLK_ECKR_bit.HSERDY != 1) {} //Ждем готовности источника такти
CLK_CKDIVR = 0;                   // Предделитель равен нулю
CLK_SWR = 0xB4;                  // Выбираем HSE источником тактов
while (CLK_SWCR_bit.SWIF != 1){}  // Ждем готовности переключения

// Включение автоматического переключения тактирования и
// разрешение прерывания по сбою HSE
CLK_CSSR = (MASK_CLK_CSSR_CSSEN + MASK_CLK_CSSR_CSSDIE);

}

int main( void ) // Основная программа
{

    init();
    asm("rim");
    while(1)          // Бесконечный цикл
    {
        asm("nop");
    }

}

// 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

    }
}

// Вектор прерывания
#pragma vector = CLK_CSS_vector
__interrupt void CLK_CSS_handler(void)
{
    // Проверка, что же вызвало прерывание
    if (CLK_CSSR_CSSD == 1)
    {
        CLK_CSSR_bit.CSSD = 0;      // Очистка флага прерывания по сб
        asm("nop");
        // Сюда можно вставить код обработки сбоя HSE
    }
}

```

And a short video demonstrating the program. In order to make the quartz fail, I touch both of its legs with my finger. You can also short the quartz legs with a

resistor Ohm to 100. When the quartz fails, you can see that the blinking frequency of the diode decreases, since STM8 clock frequency of the timer. #STM8 / EasyElectronics.ru Community

09/07/2024 17:38 from 16 MHz when working from HSE to 2 MHz when switching to HSI/8. That's



all for today, and next time we will continue to deal with timers.

Спонсор конкурса  **MASTERAM**  
МАГАЗИН ИНСТРУМЕНТОВ

STM8 , microcontrollers , programming , contest , clocking

+6

March 26, 2011, 1:32 PM

**Kalvenolt**

## Comments ( 15 )

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

Damn, I don't get it:

>> It monitors the operation of the quartz generator, and can, in case of its failure, switch the clock to the internal generator. In this case, the input frequency divider is set to 8, which corresponds to 2 MHz.

How can I make it so that it also sets the internal one to 16 MHz after detecting a malfunction of the external one? After all, all the peripherals then, when the frequency drops by 8 times, drop. I configured it for 16 MHz. I

tried to add in while(1) {} resetting the HSI prescaler to 1 (CLK\_CKDIVR\_bit.HSIDIV = 0;), but the LED still starts blinking less often... What's my mistake?

-1



**Katbert**

June 19, 2011, 00:44

I had a hard time setting the external quartz to 24 MHz. In the end I found that I need to set Options Bytes WAITESTATE = 1 wait state

0



**sergeyb2009**

September 28, 2011, 14:47

who can answer this question

I am setting up the internal high-speed RC generator 16 MHz (HSI)

**mov CLK\_DIVR, #00000000** ; setting the prescaler

; Disable switching of the clock source

**BRES CLK\_SWCR, #1** ; clear bit SWEN

;; setting the clock source

; internal high-speed RC generator 16 MHz (HSI)

**mov CLK\_SWR, #E1** ; by the way, why the value E1


if on page 108 of the RM0031 Reference manual

it is indicated **0x01** : HSI selected as system clock source (reset value) where does **E1**

0



09/07/2024, 07:38 come from then ? let's move on ; Wait for switching to be ready ; Wait for the frequency to stabilize, ; monitoring the SWIF bit in the CLK\_SWCR register **WAIT0: BTIF, CLK\_SWCR, #3, WAIT0** ; wait SWIF **BRES CLK\_SWCR, #3** ; clear SWIF and here is the main question: will the SWIF flag switch?

 **bondlab**  
November 15, 2011, 00:45

Good people, tell me...  
I'm trying to write a delay function for STM8.  
Something like this:

```
in initialization:
CLK_CKDIVR=0;
and the function itself:
void delay(unsigned int value)
{
TIM1_CR1|=(1<<7)|(1<<3); //APRE and OPM
TIM1_PSCRH=0x3e; //16000
TIM1_PSCRL=0x7f;
TIM1_ARRH=value>>8;
TIM1_ARRL=value;
TIM1_CNTRH=0;
TIM1_CNTRL=0;
TIM1_CR1|=(1<<0); //start
while ((TIM1_SR1&(1<<0))!=0); //wait for UIF
TIM1_SR1&=~(1<<0); //clear
}
```

So we're talking about the fact that the time does not converge. Calling a function with an argument of 10000 gives a delay of 3-4 seconds. I can't figure out what's going on. Please tell me...

 **urdmitriy**  
03 October 2012, 15:18

Yes, quartz is internal

 **urdmitriy**  
03 October 2012, 15:20


As far as I understand, you are setting ARPE, which means you are enabling shadow registers. Why?  
You need to call the update manually (UG bit), then reset the update flag (UIF will be set by manual update), and only then start the timer. Between UG and UIF, it is also useful to enter NOP, otherwise at high speeds the flag setting is delayed for some reason.  
I can't say more without documents.

 **angel5a**  
25 October 2012, 11:27

There is no such word as HEADER. header is read [header].

 **scaldov**  
25 October 2012, 08:57

MSP430 also has no fuses - similarly set by system flags ;) Only in most cases Value Series cannot work from HS-quartz :D

 **hex**  
October 25, 2012, 12:37

Today I decided to play around with the clocking system, but for some reason I couldn't switch the MC to an external source. The processor is stm8s003f0 (which is on Discovery),

code:

```
#include "iostm8.h"

int main( void )      // Основная программа
{
    // Настройка тактового генератора
    CLK_ECKR_bit.HSEEN = 1;          // Включаем HSE
    CLK_SWCR_bit.SWEN=1;            // Разрешаем переключение источника тактирования
    while(CLK_ECKR_bit.HSERDY != 1) {} // Ждем готовности источника тактирования
    CLK_CKDIVR = 0;                  // Предделитель равен нулю
    CLK_SWR = 0xB4;                  // Выбираем HSE источником тактовой частоты
    while (CLK_SWCR_bit.SWIF != 1){} // Ждем готовности переключения

    while(1)                      // Бесконечный цикл
    {
        // ...
    }
}
```

when debugging, the program hangs on while(CLK\_ECKR\_bit.HSERDY != 1), that is, for some reason HSE does not want to start.  
What could be the reason?



**MrMisha**

October 30, 2012, 03:50

Is the quartz connected? :)

If you are debugging not in hardware, but in an emulator (and IAR, in particular, always sets the emulator by default), then you will not get the ready bit. Look at the board, is there data exchange when you are debugging.

PS: shouldn't the SWEN bit be set after the HSERDY wait cycle? Or does it not matter?

0



**angel5a**

October 30, 2012, 1:18 PM



the problem is solved. It turned out that I forgot to solder the jumper on the discovery that connects to the external quartz))))

0



**MrMisha**

October 30, 2012, 15:47



*CSSD and CSSDIE are swapped!*

In the debugger they are also mixed up.

0



**LineAir**

November 30, 2012, 11:00

I have stm8s003f processor.

I can't get it to work from LSI!!!

I tried this:

```
CLK->ICKR |= 0x08;
```

```
CLK->SWCR |= 0x02;
```

```
while((CLK->ICKR & 16 ) != 16);
```

```
CLK->CKDIVR = 0;
```

```
CLK->SWR = 0xD2;
```

```
while ((CLK->SWCR & 0x08) != 0x08);
```

And with the help of CLK\_ClockSwitchConfig library

```
(CLK_SWITCHMODE_AUTO, CLK_SOURCE_LSI, ENABLE, CLK_CURRENTCLOCKSTATE_DISABLE);
```

It still works from HSI!

In the debugger it hangs on the loop waiting for the flag.

```
while ((CLK->SWCR & 0x08) != 0x08);
```

And in the description I came across a strange phrase:


The clock signal is not switched until the new clock source

0

 **dpochechuev**  
August 18, 2014, 11:14 PM


I'm curious about this. UART in STM8 can work both from an external quartz and from an internal RC generator. But, as far as I know, the RC generator is not as stable as quartz. Is the RC generator stable enough for UART operation?

0

 **Vitalik**  
12 February 2016, 01:13

and tell me: if all I need is to switch CKDIVR. do I need this dance with a tambourine around turning on HSI? which is already selected, but with a divider of 8. neither the docs nor the topic cover the issue :(

0

 **ku3mich**  
September 25, 2016, 01:56

Only registered and authorized users can leave comments.