

All Collective Personal TOP

Good Bad

## ADC in STM8L and everything connected with it

[STM8](#)



The STM8L15x controllers have a 12-bit ADC on board, which can operate in several modes, and supports work with a DMA controller, which allows you to digitize and put a lot of data into memory without involving the core.

Here I tried to collect as much information about the ADC in STM8 as possible, so that I wouldn't have to run around other articles looking for code to set up a timer, or, for example, DMA. Here's what's described in the article:

- Setting up the ADC
- Performing conversions in different modes
- Setting up an external trigger to start the conversion
- Setting up a timer to work with the ADC
- Using the built-in temperature sensor
- Setting up the DMA controller to work with the ADC
- Using Analog Watchdog

We won't consider the STM8S family — everything is very different there. And the STM8L101 doesn't have an ADC at all.

### To begin with, the performance characteristics of the converter:

- Programmable resolution: 6, 8, 10 or 12 bits (lowering the resolution allows for higher speed)
- Up to 28 external channels. But the MCU that is based on the STM8L-Discovery has "only" 25 of them.
- Plus two more internal channels: a temperature sensor and a reference voltage.
- Two main operating modes:
  - 1) Single conversion
  - 2) Sequential reading of several channels, with data transfer to the buffer using DMA. In this case, the MCU is not distracted by switching channels or writing data to the buffer.
- External and internal triggers.
- Either the Vdda pin or the Vref+ pin acts as a reference voltage (in this case, the voltage on it should be > 2.4V). In the Discovery, VREF is tightly nailed to the power supply.

## Setting

### 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](#)

Before starting work, it would be a good idea to find everything connected with it. For example, with other peripherals, the ADC needs to be clocked. The very first bit in the CLK\_PCKENR2 register is responsible for this:

```
CLK->PCKENR2 |= CLK_PCKENR2_ADC1; //Подает тактирование на АЦП
```

Initially, the ADC is in sleep mode. To wake it up, you need to raise the ADON bit in the ADC\_CR1 register. But you can't start converting right away — you need to wait about 3  $\mu$ s until the converter wakes up.

If the ADC is idle for some time, it will go into sleep mode itself. This “some time” — Tidle — is usually 1 sec, but can decrease sharply with increasing temperature.

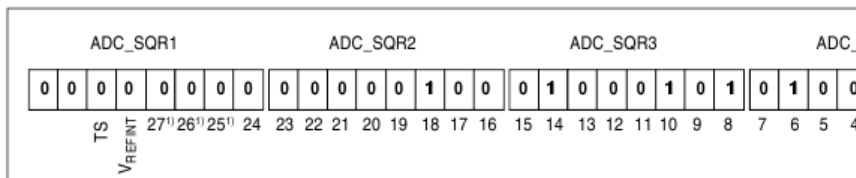
After the ADC wakes up, you need to select the channels from which you want to digitize the signal. You can select several channels at once through the ADC\_SQR1 — ADC\_SQR4 registers. The CHSEL\_Sx bits in them are responsible for each channel. To select a channel, simply set the corresponding bit (if you set several, they will be processed starting with the least significant). The CHSEL\_STS and CHSEL\_SVREFINT bits are responsible for selecting the temperature sensor and reference voltage, respectively.

In the header file `stm8l15x.h`, the `ADC_SQRx` registers are organized into an array, so the numbering starts from 0. For example, like this

```
ADC1->SQR[3] |= (1<<4);
```

we will set the 4th bit in the ADC\_SQR4 register.

For example, if you set the bits like this, the following sequence of channels will be processed: 0, 2, 6, 8, 10, 14, 18.



In the **TRIGR1** — **TRIGR4** registers (which are also linked into an array, so the numbering starts from zero), each bit is responsible for disabling the Schmitt trigger on the corresponding channel. When writing to bit 1, the trigger is disabled and the pin cannot be used as a digital input. But if you do not disable them, the accuracy will be an order of magnitude worse.

For example, when measuring the voltage from a potentiometer with the trigger enabled, the results could differ by several dozen LSBs. When the trigger is disabled, this difference is reduced to a couple of LSBs.

Note that the DMAOFF bit is hidden in the ADC\_SQR1 register, which is responsible for the operation of the ADC together with the DMA. By default, it is reset, which means the DMA controller is enabled. If you don't need DMA, then write 1 to this bit. **The SMP1[2:0]**

bits in the ADC\_CR2 register control the ADC sampling time for the first 24 channels. For the rest (in Discovery, this is the last external channel and two internal ones), there are the SMP2[2:0] bits in ADC\_CR3. The SMPx[2:0] bits can take the following values: **000** — Sampling time = 4 ADC cycles **001** — 9 cycles **010** — 16 cycles **011** — 24 cycles **100** — 48 cycles **101** — 96 cycles **110**

[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](#)

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

[All blogs](#)

— 192 cycles **111** — 384 cycles The conversion time largely depends on this parameter. But don't rush to set the ADC in STM8L and everything is connected with high resistance at the input, then you need to set a long sampling time so that the capacitors have time to charge. More information about the error caused by external resistance can be found in the application note titled "[A/D converter on STM8L devices: description and precision improvement techniques](#)" in the section "External resistance design error" If there is no need to use all 12 bits, the ADC resolution can be reduced. For this, there are RES[1:0] bits in the ADC\_CR1 register: **00** — 12 bits **01** — 10 bits **10** — 8 bits **11** — 6 bits The ADC speed depends on the resolution and sampling time. The time required for one conversion is calculated using the following formula:  **$T_{conv} = (ts+n)/f_{ADC}$**  From this, it is easy to obtain **the maximum ADC speed :  $ADCSpeed = f_{ADC}/(ts+n)$**  In these formulas, **fADC** is the clock signal frequency on the ADC **ts** is the sampling time (in cycles) **n** is the ADC resolution

For example, with a clock rate of 8 MHz, a resolution of 10 bits and a minimum sampling time (4 cycles), we can get a speed of 571.4 kHz. If for some reason it is necessary to reduce the speed, then you can set the PRESC bit in the ADC\_CR2 register. This will make the converter work twice as slowly.

For example, here is **the simplest ADC initialization** :

```
CLK->PCKENR2 |= CLK_PCKENR2_ADC1; //Тактирование

ADC1->CR1 |= ADC_CR1_ADON; //Пинаем АЦП, чтобы он проснулся

ADC1->SQR[0] |= ADC_SQR1_DMAOFF; //Отключаем DMA
ADC1->SQR[3] |= 1<<2; //Выбираем 2 канал (пин A4 на STM8L152C6T6)
```

After executing this code, the ADC will be set to 12-bit resolution and minimum

We have learned how to set up the ADC, now we want to measure and convert something.

## Single conversion mode

The simplest mode, in which the ADC reads the voltage from one channel after starting and stops. If more than one channel is selected in the ADC\_SQRx registers, the one with the higher number will be processed.

Before conversion, it is necessary (in addition to the usual ADC setup, see above) to disable DMA. To do this, set the DMAOFF bit in the ADC\_SQR1 register.

**To start the conversion**, set the START bit in the ADC\_CR1 register. As soon as it is completed, the EOC flag in ADC\_SR will rise. It is by this that you need to determine the completion of the conversion, because the START bit is reset immediately after the start. The EOC bit is reset itself after reading the conversion result (more precisely, after reading the ADC\_DRH register).

If the **EOCIE** bit is set in the ADC\_CR1 register, an interrupt will occur upon completion of the conversion. When using an interrupt, it is necessary to read the ADC\_DRH register in its handler, otherwise the interrupt flag will not be reset and, after exiting the handler, the controller will return to it again.

**The result is stored in the ADC\_DRH and ADC\_DRL registers**, right-aligned.

| Разрешение      | ADC_DRH  | ADC_DRL         |                 |                 |                 |                 |                 |                 |                 |  |                 |                 |                 |                 |                |                |                |                |
|-----------------|--|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|--|-----------------|-----------------|-----------------|-----------------|----------------|----------------|----------------|----------------|
| 12 бит          | <table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>D<sub>11</sub></td><td>D<sub>10</sub></td><td>D<sub>9</sub></td><td>D<sub>8</sub></td></tr></table> | 0               | 0               | 0               | 0               | D <sub>11</sub> | D <sub>10</sub> | D <sub>9</sub>  | D <sub>8</sub>  | <table><tr><td>D<sub>7</sub></td><td>D<sub>6</sub></td><td>D<sub>5</sub></td><td>D<sub>4</sub></td><td>D<sub>3</sub></td><td>D<sub>2</sub></td><td>D<sub>1</sub></td><td>D<sub>0</sub></td></tr></table>   | D <sub>7</sub>  | D <sub>6</sub>  | D <sub>5</sub>  | D <sub>4</sub>  | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
| 0               | 0  | 0               | 0               | D <sub>11</sub> | D <sub>10</sub> | D <sub>9</sub>  | D <sub>8</sub>  |                 |                 |  |                 |                 |                 |                 |                |                |                |                |
| D <sub>7</sub>  | D <sub>6</sub>   | D <sub>5</sub>  | D <sub>4</sub>  | D <sub>3</sub>  | D <sub>2</sub>  | D <sub>1</sub>  | D <sub>0</sub>  |                 |                 |  |                 |                 |                 |                 |                |                |                |                |
| 10 бит          | <table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>D<sub>11</sub></td><td>D<sub>10</sub></td></tr></table>                         | 0               | 0               | 0               | 0               | 0               | 0               | D <sub>11</sub> | D <sub>10</sub> | <table><tr><td>D<sub>9</sub></td><td>D<sub>8</sub></td><td>D<sub>7</sub></td><td>D<sub>6</sub></td><td>D<sub>5</sub></td><td>D<sub>4</sub></td><td>D<sub>3</sub></td><td>D<sub>2</sub></td></tr></table>   | D <sub>9</sub>  | D <sub>8</sub>  | D <sub>7</sub>  | D <sub>6</sub>  | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> |
| 0               | 0  | 0               | 0               | 0               | 0               | D <sub>11</sub> | D <sub>10</sub> |                 |                 |  |                 |                 |                 |                 |                |                |                |                |
| D <sub>9</sub>  | D <sub>8</sub>   | D <sub>7</sub>  | D <sub>6</sub>  | D <sub>5</sub>  | D <sub>4</sub>  | D <sub>3</sub>  | D <sub>2</sub>  |                 |                 |  |                 |                 |                 |                 |                |                |                |                |
| 8 бит           | <table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>   | 0               | 0               | 0               | 0               | 0               | 0               | 0               | 0               | <table><tr><td>D<sub>11</sub></td><td>D<sub>10</sub></td><td>D<sub>9</sub></td><td>D<sub>8</sub></td><td>D<sub>7</sub></td><td>D<sub>6</sub></td><td>D<sub>5</sub></td><td>D<sub>4</sub></td></tr></table> | D <sub>11</sub> | D <sub>10</sub> | D <sub>9</sub>  | D <sub>8</sub>  | D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> |
| 0               | 0  | 0               | 0               | 0               | 0               | 0               | 0               |                 |                 |  |                 |                 |                 |                 |                |                |                |                |
| D <sub>11</sub> | D <sub>10</sub>  | D <sub>9</sub>  | D <sub>8</sub>  | D <sub>7</sub>  | D <sub>6</sub>  | D <sub>5</sub>  | D <sub>4</sub>  |                 |                 |  |                 |                 |                 |                 |                |                |                |                |
| 6 бит           | <table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>   | 0               | 0               | 0               | 0               | 0               | 0               | 0               | 0               | <table><tr><td>0</td><td>0</td><td>D<sub>11</sub></td><td>D<sub>10</sub></td><td>D<sub>9</sub></td><td>D<sub>8</sub></td><td>D<sub>7</sub></td><td>D<sub>6</sub></td></tr></table>                         | 0               | 0               | D <sub>11</sub> | D <sub>10</sub> | D <sub>9</sub> | D <sub>8</sub> | D <sub>7</sub> | D <sub>6</sub> |
| 0               | 0  | 0               | 0               | 0               | 0               | 0               | 0               |                 |                 |  |                 |                 |                 |                 |                |                |                |                |
| 0               | 0  | D <sub>11</sub> | D <sub>10</sub> | D <sub>9</sub>  | D <sub>8</sub>  | D <sub>7</sub>  | D <sub>6</sub>  |                 |                 |  |                 |                 |                 |                 |                |                |                |                |

**You should read the high register first, and then the low register .**

Although if the resolution is set to 6 or 8 bits, then it is not necessary to read the high register.

This is how you can start the conversion and read the result into the result variable (uint16):  
(before this, the ADC must be configured correctly (see above))

```
ADC1->CR1 |= ADC_CR1_START; //Поехали!

while ((ADC1->SR && ADC_SR_EOC) == 0); //ждем, пока выполнится преобразо

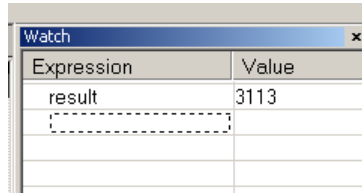
//Забираем результат
result = ADC1->DRH << 8;
result |= ADC1->DRL;
```

The archive contains [an example for IAR](#), which measures the voltage at the A4 input.

Usually in such examples the measurement result is output to UART or to the indicator.

But I decided to do it differently - to use the debugger built into STM8L-Discovery for these purposes.

In the example, the result is simply read into the result variable, the value of which can be observed through the debugger:



| Expression | Value |
|------------|-------|
| result     | 3113  |
|            |       |
|            |       |
|            |       |

Some may think that this is inconvenient, but the program is freed from unnecessary code.

## Continuous operation mode

Setting the CONT bit in the ADC\_CR1 register starts the ADC in the continuous conversion mode. It reads the values of all selected channels, starting with the least significant. As soon as it reaches the last channel, it can generate an EOC interrupt and start from the beginning.

In this mode, it is necessary to use the DMA controller, because the EOC interrupt occurs only when the conversion of all selected channels is completed, and it is impossible to read the result from the ADC\_DRH and ADC\_DRL registers.

If the ADC starts from an external trigger, then after starting in the continuous conversion mode, it will perform conversions until it is stopped. This can be done in two ways: reset the CONT bit in the ADC\_CR1 register - the ADC will stop when it completes the conversion of the current chain of channels; or reset the ADON bit in the same ADC\_CR1, thereby immediately turning off the converter. It

is also possible to enable operation via DMA (and configure it correctly, which will be discussed later) without setting the CONT bit. In this case, the ADC will convert all selected channels, DMA will write the result to RAM, after which the ADC will stop.

## Starting a transformation from external triggers

The ADC can be started not only via the START flag, but also from external (relative to the converter) triggers. In STM8L15x there are three of them:

- **From pins A6 or D0 (TRIG1)**
- **By event from timer 1 (TRIG2)**
- **By event from timer 2 (TRIG3)** The **EXTSEL1** and **EXTSEL0** bits in ADC\_CR2

are used to select the trigger. They can take the following values: **00** — Software start **01** — TRIG1 (pin A6 or D0) **10** — TRIG2 (Timer 1) **11** — TRIG3 (Timer 2) The edge of the trigger signal, which will start the ADC, is selected by the **TRIG\_EDGE1** and **TRIG\_EDGE0** bits in the same ADC\_CR2: **00** — Reserved **01** — Rising edge **10** — Falling edge **11** — Both edges For 1 trigger, this means the rising (rising) or falling (descending) edge on the selected pin. For the other two, this is the edge of the signal of the corresponding event (specified when configuring the timer). For them, it is most logical to select the rising edge — then the ADC will start immediately as the event occurs.

## TRIG1 - Signal from pin

The first trigger can switch between pins A6 and D0. By default, it is set to A6. To switch to D0, you need to set the **ADC1TRIG\_REMAP** bit, which is located in the SYSCFG\_RMPCR2 register.

No additional configuration of the pin from which the start signal will be sent to the ADC is required. This code **will configure the ADC to start from a voltage drop on pin A6** :

```
GPIOA->CR1 |= GPIO_Pin_6; //Подтягивающий резистор на пин A6 (не обязае.

CLK->PCKENR2 |= CLK_PCKENR2_ADC1; //Тактирование

ADC1->CR1 |= ADC_CR1_ADON; //Будим АЦП
ADC1->CR2 |= (1<<6) | (1<<3); //Старт по заднему фронту от 1 триггера.

ADC1->SQR[0] |= ADC_SQR1_DMAOFF; //Отключаем DMA
ADC1->SQR[3] |= 1<<2; //Выбираем 2 канал (пин A4 на STM8L152C6T6)
```

Now, as soon as the voltage on pin A6 drops to log 0, the conversion will start.

## TRIG2, TRIG3 — Timers

The first or second timer can be used as a source of the start signal for the ADC. Each of them has a TRGO (Trigger output), which is connected to the ADC trigger input inside the crystal. Timers can pull this output for various reasons - counter overflow, counter coincidence with the comparison register, etc. The ADC receives the signal and begins the conversion.

We will consider the simplest mode, in which **the timer counts from 0 to some value, upon reaching which it gives a signal to the ADC and resets again**. Thus, the ADC will perform conversions at the time intervals we need.

The timer setup for our purposes looks like this:

```
CLK->PCKENR1 |= CLK_PCKENR1_TIM2; //Подаем тактирование на таймер.

//Волшебное число 46875 получилось так:
// Частота МК (2М) / Предделитель таймера (128) * Нужное время в секун
TIM2->ARRH = 46875>>8; //Записываем сначала старший байт
TIM2->ARRL = (uint8_t)(46875); //Потом младший
TIM2->CR1 |= TIM_CR1_URS; //Настраиваем источник события update
TIM2->CR2 |= (2<<4); //Настраиваем на подачу сигнала для АЦП при переп
TIM2->PSCR = 7; //Предделитель = 2^7 = 128
TIM2->EGR |= TIM_EGR_UG; //Генерим событие update, чтобы обновился пр
TIM2->CR1 |= TIM_CR1_CEN; //Запускаем
```

I think explanations are unnecessary here. All the most important things are

indicated in the comments to the code.

09/07/2024, 07:37

ADC in STM8L and everything connected with it / STM8 / EasyElectronics.ru Community

If something is still unclear, [there is an example](#) in which the timer sends a signal to the ADC every 500 ms. In the interrupt, upon completion of the conversion, the blue LED switches - so that it is visible how often the ADC is triggered.

## Using the built-in temperature sensor

The STM8L has its own temperature sensor connected to the **ADC\_TS** channel in the ADC. Of course, it is not particularly accurate, but it can perform measurements of the "heat/cold" type.

Before reading the sensor readings, it must be started (initially turned off to save electricity). The TSON bit in the ADC\_TRIG1 register is used to control the sensor power supply. After it is set, **you must wait 10 microseconds** for the sensor to turn on.

It is recommended to set the sampling time (for the temperature sensor, it is set by the SMTP2[2:0] bits) to **more than 10 microseconds**. If you set it less, the high output resistance of the sensor will distort the result. Or it may not, but you must follow the recommendations.

Apart from starting, working with the sensor is no different from working with any other channel.

The code that starts the temperature sensor and reads its readings into the result variable:

```
CLK->PCKENR2 |= CLK_PCKENR2_ADC1; //Тактирование

ADC1->CR1 |= ADC_CR1_ADON;
ADC1->TRIGR[0] |= ADC_TRIGR1_TSON; //Подаем питание на датчик

ADC1->CR2 |= 3; //Sampling time = 12uS
ADC1->SQR[0] |= ADC_SQR1_DMAOFF | (1<<5); //Отключаем DMA и выбираем датчик

delay_10us(1); //Функция из delay.h

ADC1->CR1 |= ADC_CR1_START; //Начинаем измерение

while ((ADC1->SR && ADC_SR_EOC) == 0); //Ждем, пока выполнится преобразование

//Забираем результат
result = ADC1->DRH << 8;
result |= ADC1->DRL;
```

**Testing this sensor in practice** has shown that it has quite large (about 10 degrees) deviations at high and low temperatures. At room temperature, the sensor accuracy is slightly higher.

## Measuring reference voltage

The STM8L has a built-in reference voltage, which is not surprising. What is surprising is that it cannot be used as a reference voltage for the ADC. But the reference voltage can be measured using the ADC. This is necessary, for example, to calculate the supply voltage of the microcontroller. Or to make a correction in the measured values (with an unstable external reference).

The reference voltage in the STM8L15x **is about 1.224V** (minimum - 1.202,

```
#define Factory_VREFINT 0x4910
```

And then we read as a normal register:

```
uint16_t ref;  
...  
ref = 0x600 + Factory_VREFINT;
```

The high byte of this value is always 0x06, and the low byte is located at the address VREFINT\_Factory\_CONV.

In order for the reference voltage to be measured via the ADC, the **VREFINTON bit** must be set in the ADC\_TRIGR1 register.

After that, the voltage can be measured **via the VREFINT channel** (the fifth bit in ADC\_SQR1).

Using the reference voltage value, you can correct the conversion result:

$$V_{IN} = \frac{D_{IN}}{D_{REFINT}} \times V_{REFINT}$$

In this formula,

**Din** is the measured voltage

**Vrefint** is the reference voltage value in volts. Taken from the datasheet or measured earlier.

**Drefint** is the result of measuring the reference voltage.

Replacing Din in this formula with the maximum value for a given resolution (1023 for 10 bits, 4095 for 12 bits ...), we get the ADC reference voltage value as a result. And if it is connected to the Avcc pin, then this way **you can determine the supply voltage quite accurately**.

This can be useful in the case of a device that is powered by batteries without any converters. Or, if the device can be powered by several sources with different voltages - you can determine from which source the device is currently powered.

## Using DMA

There is often a need **to collect a bunch of samples from the ADC into RAM**. For this, you can use the DMA controller. Unlike manual data transfer to RAM, here we only need to configure the DMA correctly and start the ADC.

All the DMA does is that when the conversion is complete, it takes data from the **ADC\_DRH** and **ADC\_DRL** registers, writes them to the buffer at the specified address, and increments this address. At the same time, the DMA controller counts the number of bytes that have already been written. As soon as the required number has been written, an interrupt occurs.



```
uint16_t recv_array[4];
```

The names of the DMA bits and registers in the STM8L15x.h header file correspond to the datasheet slightly less than not at all. It is unclear why ST engineers did this, but one can only guess the purpose of a particular bit based on the description.

Now we set up DMA:

```
CLK->PCKENR2 |= CLK_PCKENR2_DMA1; //Подает тактирование

DMA1_Channel0->CNTR = sizeof(recv_array)-1; //Размер буфера

DMA1_Channel0->CPARH = (ADC1_BASE+4)>>8; //Адрес регистра АЦП (старший б
DMA1_Channel0->CPARL = (uint8_t)(ADC1_BASE+4); //Младший

DMA1_Channel0->CM0ARH = (uint8_t)((uint16_t)recv_array>>8); //Адрес бу
DMA1_Channel0->CM0ARL = (uint8_t)recv_array;

DMA1_Channel0->CSPR |= DMA_CSPR_16BM; //Режим работы с 16и битными чи
DMA1_Channel0->CCR |= DMA_CCR_IDM | DMA_CCR_CE | DMA_CCR_TCIE; //Включ
DMA1->GCSR |= DMA_GCSR_GE; //Включаем DMA
```

After starting the ADC, the DMA will add the conversion results to the recv\_array array. When it is full, the Transaction Complete interrupt will be triggered. **If the interrupt is not needed**, the DMA\_CCR\_TCIE bit should be cleared.

**If the ADC bit depth is 8 or 6 bits** (i.e. the result is placed in the ADC\_DRL register), then the line:

```
DMA1_Channel0->CSPR |= DMA_CSPR_16BM; //Режим работы с 16и битными числ
```

It can be removed from the setup procedure. And when specifying the register address,

```
DMA1_Channel0->CPARH = (ADC1_BASE+4)>>8; //Адрес регистра АЦП (старший б
DMA1_Channel0->CPARL = (uint8_t)(ADC1_BASE+4); //Младший
```

replace the number 4 with 5. Then the DMA will read 1 byte from the ADC\_DRL register (which has an offset of 5 relative to the ADC base address) and write it to the array.

You can also set **the CIRC bit in the CCR register**. In this case, when the buffer is full, the DMA will not stop, but will start filling it from the beginning, overwriting the old data

```
DMA1_Channel0->CCR |= DMA_CCR_ARM; //ARM - Auto-Reload mode
```

In this case, you can disable interrupts and enjoy the fact that the buffer

It should be noted that if the continuous conversion mode is enabled (the CONT bit), but the CIRC bit is not set, then when the buffer is full, DMA will stop, but the ADC will continue to work. The data will not be saved anywhere.

In [this example](#), using DMA, data from 4 channels (1,2,3,4) is added to an array in RAM.

The channels correspond to the following pins:

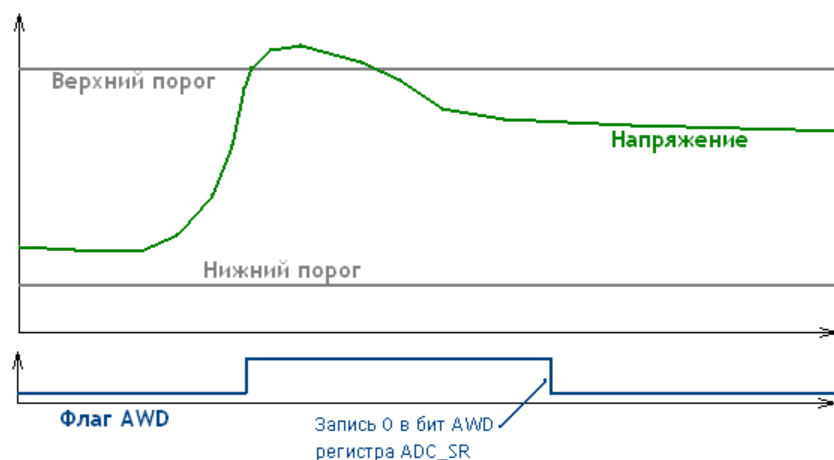
- 1 - A5
- 2 - A4
- 3 - C7
- 4 - C4

## Analog Watchdog

The ADC has one useful feature — **voltage monitoring on a channel**.

It works like this:

After the conversion is complete, the current channel number is compared with the one to be monitored. If they match, the conversion result is compared with two thresholds — lower and upper. If the result is greater than the upper threshold or less than the lower threshold, the **AWD flag** in the ADC\_SR register is raised. **You can set the AWDIE bit in ADC\_CR1, then the analog watchdog will be able to call an interrupt**.



**The AWD flag must be lowered manually** by writing 0 to it. This creates a problem: we can catch the moment when the voltage goes beyond the set limits, but whether it has returned back to these limits will have to be checked manually.

**The channel that the analog watchdog will monitor is selected via the CHSEL[4:0] bits in the ADC\_CR3 register**. The value 0 corresponds to channel zero, 1 — to channel one, etc. The upper threshold is configured via the ADC\_HTRH and ADC\_HTRL registers, and the lower threshold is configured via ADC\_LTRH and ADC\_LTRL.

The threshold values are written taking into account the current ADC resolution. For example, 1/4 of the reference voltage is 64 for an 8-bit resolution, or 1024 for a 12-bit resolution.

The thresholds must be configured and the channel selected before starting the ADC.

This is how you can configure Analog Watchdog (it is assumed that the ADC has

already been configured):

09/07/2024, 07:37

ADC in STM8L and everything connected with it / STM8 / EasyElectronics.ru Community

```
ADC1->CR1 |= ADC_CR1_AWDIE; //Разрешаем прерывание AWD

//Устанавливаем верхний (4000) и нижний (100) пороги
#define AWD_low_th 100
#define AWD_high_th 4000

ADC1->HTRH = AWD_high_th>>8;
ADC1->HTRL = (uint8_t)(AWD_high_th);

ADC1->LTRH = AWD_low_th>>8;
ADC1->LTRL = (uint8_t)(AWD_low_th);

ADC1->CR3 &= ~ADC_CR3_CHSEL; //Сбрасываем канал
ADC1->CR3 |= 2; // и выбираем канал №2 для отслеживания
```

Now you can either **monitor the AWD bit in ADC\_SR** (if the interrupt is not used),

```
if ((ADC1->SR & ADC_SR_AWD) != 0)
{
    . . .
}
```

or wait for an interrupt. In its handler, you will need to **reset the AWD bit** by writing 0 to it:

```
INTERRUPT_HANDLER(ADC_IRQ, 18)
{
    if ((ADC1->SR & ADC_SR_AWD) != 0)
    {
        . . .
        ADC1->SR &= ~ADC_SR_AWD;
    }
}
```

**That's all.** Working with ADC in all possible modes is described, examples for a quick start and "for playing around" are available. One question remains - next time describe the subject in the same detail, or is a short description enough, as in [the article about LCD](#) ?



STM8L , STM8 , ADC , competition2

+8

07 September 2011, 23:32

**dcoder**

3

Files in the topic: [ADC-3-DMA.zip](#) , [ADC-2-External Trigger.zip](#) , [ADC-1-Simple conversion.zip](#)

## Comments ( 21 )

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

In detail, in detail. Aren't you putting it up for competition? ;)


0



**DIHALT**

07 September 2011, 23:43

0

 **dcoder**  
07 September 2011, 23:48

1


Powerfully pushed, inspires :). I wish to continue in the same spirit, it is very interesting to read a detailed analysis.

0

 **DareDen**  
07 September 2011, 23:46

As always, your articles are a joy. Thank you.

0

 **Episcop**  
07 September 2011, 23:49

Credit!

0

 **hellraiser**  
08 September 2011, 00:24


Great, keep it up!

0

 **bdpcvit**  
08 September 2011, 16:35


Great article! Thanks.

0

 **uschema**  
08 September 2011, 17:43

only details, only hardcore!  
The article is just delicious, wild respect!

0

 **berrymorr**  
09 September 2011, 09:35

*But you can't start the conversion right away – you need to wait about 3 μs until the converter wakes up.*

Isn't there some kind of readiness flag? And anyway, is there any way to determine if the ADC has quietly fallen asleep?


0

 **Vga**  
09 September 2011, 10:54

No, that's the point. Here's what RM0031 writes:

0


*The time between 2 conversions must be lower than the ADC maximum idle delay (tIDLE). In case the time between 2 conversions is greater than tIDLE, the ADC must be powered-off between the 2 conversions (by clearing the ADON bit). Another possibility is to discard the first conversion (occurring in a time greater than a tidle after the previous one) and keep the next one*

 **dcoder**  
09 September 2011, 11:01

1

I apologize in advance if the question is lame. I tried the example with the timer. I applied a constant voltage to the pin. It shows the required value stably, and then drops almost to zero, then it is normal again, then it gives much more (almost 4000). And so on with a period of 4-5 measurements... Can you tell me what the problem could be?

0


 **Shadow**  
November 27, 2011, 10:48 PM

How do you collect data? Do you write it to memory or transmit it via UART?

0

I have stm8l-discovery. I tried to output data both to the LCD in real time, and in the debugger (IAR) I set a breakpoint and looked at the result value. Maybe I just made a mistake in the circuit. I connected Vcc to GND through two identical resistors in series, and connected port A4 between them. In theory, there should be ~ 2048 on the port. That's what it turns out to be, but sometimes it flies to 440 or rises to 3600.

0

 **Shadow**  
November 28, 2011, 09:08


And by chance I define the address of the Factory\_VREFINT register:

```
#define Factory_VREFINT 0x4910
```

0

Isn't it supposed to be like this?

```
#define Factory_VREFINT ((uint8_t *)0x4910)
```

 **anar**  
November 28, 2011, 03:23

Of course you are right. Yesterday, while working on the ADC in STM8L051F3, I spent a long time looking for the address of this cell, it is not in the datasheets for the chip, I found it only here, thanks to the author. I copied it without thinking, but no. I came to the same conclusion as you, and it worked. The supply voltage is determined quite accurately, without using a single pin (which is important for a chip with 20 legs), only by measuring the internal support. By the way, the datasheet at least states that calibration is performed at a supply voltage of 3.0 V, at first I thought that at 3.3.

0


 **dadigor**  
09 December 2018, 04:39

I fought with ADC+DMA for a long time - one-time data collection in order to use two different buffers. As it turned out, you need to cleverly handle the interrupt from DMA so that it starts working again:

0

```
#pragma vector=DMA1_CH0_CH1_vector
__interrupt void DMA_Handler(void)
{
    DMA1_GCSR &= ~(0x01);
    DMA1_C0SPR &= ~(0x02|0x04);
    DMA1_C0NDTR = sizeof(recv_array)-1;
    DMA1_GCSR |= 0x01;
    //DMA1_C0CR &= ~(0x01);
    //DMA1_C0CR |= 0x01;
    ADC1_CR1 |= 0x02;
}
```

Of course, maybe I don't have a good understanding of all the intricacies of the process, but maybe it will be useful to someone?!

 **SpiritWar**  
April 23, 2012, 14:05

*The result is stored in the ADC\_DRH and ADC\_DRL registers, right-aligned.  
The higher register should be read first, then the lower one. Although if the resolution is set to 6 or 8 bits, then it is not necessary to read the higher one.*

0


I haven't tested it on STM8L, I'll take your word for it.

What's interesting is that the Family Guide to STM8S says the following:

**24.8 Data alignment**  
<...>  
*Right Alignment: 8 Least Significant bits are written in the ADC\_DL register, then the remaining Most Significant bits are written in the ADC\_DH register. The Least Significant Byte must be read first followed by the Most Significant Byte. <...>*


It says here that with right alignment, the low byte should be read first, and then the high byte.

And if you break this order, then you really get irrelevant results, apparently after raising the

 **akaChewy**  
June 25, 2013, 08:33



Thank you very much for the manual — everything worked in an evening on my knee. But there is one problem — DMA and ADC do not work synchronously (. Is it possible to describe in a little more detail who calls whom and who waits or does not wait for whom?  
I wanted to make a constant poll of 4 analog channels with a call to the DMA  
DataTransferComplete interrupt after each poll of a set of analog inputs. Everything is fine, but when debugging, data gets to the buffer starting from a random input from a group (for example, 1234,3412,2341,3412, etc.) — apparently, the ADC continues to hammer without looking back at the DMA. Or maybe the debug mode stops only the core, and the DMA and ADC live their own lives?  
  
For debugging, I tried to turn off CONT for the ADC and / or ARM for the DMA - so the DMA interrupt simply stopped being called. But the buffer is filled with data once.

0

 **executer**  
06 April 2014, 22:20


As it happens, desynchronization occurred when exiting to break during debugging. The first break shows everything well, even several minutes after the reset. On subsequent breaks, the data shifts. [roboforum.ru/forum59/topic14142.html](http://roboforum.ru/forum59/topic14142.html)

0

 **executer**  
April 11, 2014, 02:40 


I am making a very cheap device on stm8L. I had to switch from Discovery to a cheaper STM8L051F3 processor, I couldn't find calibration values for the ADC in the datasheet. Factory conversion registers — there is no such section. Are they not calibrated at all? 1202mV-1242mV — somehow a very large spread. Who knows, maybe they are calibrated after all, but for some reason the registers were not specified?

0

 **dpochechuev**  
September 24, 2018, 5:27 PM

Since I encountered STM8 for the first time, after checking, I am writing so that others do not get caught by the wrong part of the code:  
for the correct operation of the DMA mode with the ADC, you need to use  
DMA1\_Channel0->CNBTR = sizeof(recv\_array)-1; //Buffer size instead of DMA1\_Channel0->CNBTR = sizeof(recv\_array)>>1; //Buffer size which will correct the above-mentioned faults of type 1234, 2341,... and will not spoil the program. (according to the documentation, this address specifies the amount of information transferred in bytes or words, depending on the specified DMA\_CSPR\_16BM mode) I ask the author to correct this.

0

 **ChipLeo**  
03 August 2019, 12:03

Only registered and authorized users can leave comments.