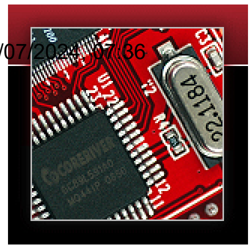

Working with EEPROM and Flash

STM8

All STM8 MCUs are equipped with some amount **of EEPROM** a. It varies from a measly 128 bytes in the junior STM8S models to 2 KB in the senior STM8L and S models. Thanks to the single address space, working with EEPROM is almost no different from working with RAM.

Not only EERPOM, but also **flash** is written easily and simply. Therefore, although the note is devoted to working with EEPROM, there will be some comments regarding flash.

In **the STM8L152C6** (which is on the Discovery), the EEPROM is located starting from address **0x1000** . It occupies 1 kilobyte, extending to address **0x13FF** . Flash (program memory) starts from address **0x8000** .

Reading from EEPROM is similar to reading from any other memory area - Flash, RAM or peripheral registers.

Writing data is also similar to writing to other memory areas. In addition to simply writing 1 byte, it is possible to write 4 bytes in one go , which, given the long write time in EEPROM (6ms), can be very useful. Moreover, an entire block (128 bytes) can be written in one go , but for this, the function that will perform the write must be placed in RAM.

The standard library provides us with several functions for working with EEPROM (and flash) memory. All of them are in the **stm8l15x_flash.h** module . The first one implements reading:

```
uint8_t FLASH_ReadByte(uint32_t Address);
```

As the name suggests, the function reads a byte at the specified address. But if we need to read not a byte, but, for example, a word or a double word, then it is easier to do this:

```
#pragma location=0x1000 //Наш адрес в EEPROM/flash (в данном случае - на
__no_init uint32_t MyEEPROMVariable
```

The location directive tells the compiler where to place the variable, and

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 __*no_init* prevents it from being reset at startup.

Now we can work with *MyEEPROMVariable* as with a regular variable. Copy it to another variable, compare it, write it, etc. Although no, we cannot write to it yet, because the memory is write-protected.

In order to write, the EEPROM must be unlocked. The lock is enabled at MCU startup and serves to protect against unexpected program actions or failures. The protection is removed by sequentially writing two keys to the **FLASH_DUKR** register :

```
FLASH_DUKR = 0xAE;
FLASH_DUKR = 0x56;
```

To unlock the flash, use another magic trick:

```
FLASH_PUKR = 0x56;
FLASH_PUKR = 0xAE;
```

The same can be done using a function from the library:

```
FLASH_Unlock(FLASH_MemType_Data); //Разблокировка EEPROM
FLASH_Unlock(FLASH_MemType_Program); //Разблокировка памяти программ
```

Now we can write to our *MyEEPROMVariable* . It should be noted that when writing to program memory, the core will stop for the entire duration of the write, and when writing to EEPROM, it will work quietly while the byte is being written.
To write to a specific address, the library has the functions **FLASH_ProgramByte** (uint32_t Address, uint8_t Data) and **FLASH_ProgramWord** (uint32_t Address, uint32_t Data).

The first one is nothing interesting, but the second one uses the mechanism of writing a double word in one go.

```
FLASH->CR2 |= FLASH_CR2_WPRG;

 /* Write one byte - from lowest address*/
  *((PointerAttr uint8_t*)(uint16_t)Address) = *((uint8_t*)(&Data));
 /* Write one byte*/
  *(((PointerAttr uint8_t*)(uint16_t)Address) + 1) = *((uint8_t*)(&Data)
 /* Write one byte*/
  *(((PointerAttr uint8_t*)(uint16_t)Address) + 2) = *((uint8_t*)(&Data)
 /* Write one byte - from higher address*/
  *(((PointerAttr uint8_t*)(uint16_t)Address) + 3) = *((uint8_t*)(&Data)
```

First, it sets the **WPRG** bit in the FLASH_CR2 register, then writes 4 bytes in a row to the memory. In this case, the write operation does not occur, and the bytes end up in the buffer. After writing the last byte , the entire batch is automatically written to the memory. Thus, the overall write speed increases by 4 times.

Let's try to speed up the write process even more by pushing the entire block (128 bytes) at a time. For this, the **FLASH_ProgramBlock** function is used (uint16_t BlockNum, FLASH_MemType_TypeDef FLASH_MemType, FLASH_ProgramMode_TypeDef FLASH_ProgMode, uint8_t* Buffer)

BlockNum is the number of the block to be written. For example, block 0 is located at addresses 0x1000 - 0x107F.

FLASH_MemType is the memory to which we want to write the data. *FLASH_MemType_Data* for writing to EEPROM or *FLASH_MemType_Program* for

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

writing to program memory (flash).

FLASH_ProgMode — writing mode. It can be standard (*FLASH_ProgramMode_Standart*), in which writing takes 6 milliseconds, or fast (*FLASH_ProgramMode_Fast*) — writing will be done twice as fast, but it is necessary to clear the block first.

Buffer — a pointer to an array the size of one block (for example, 128 bytes).

But you won't be able to use this function just like that. For it to work, **FLASH_ProgramBlock** must be executed from RAM . Fortunately, it is very easy to implement — the library developers and IAR took care of everything. All we have to do is uncomment the line

```
#define RAM_EXECUTION  (1)
```

in **the stm8l15x.h** file . Now those functions that should be executed from RAM will be copied to it at startup.

The block writing mechanism itself works similarly to the double word writing mechanism. At the beginning, the **PRG** bit is set (or **FPRG** if the fast writing mode is used) in the **FLASH_CR2** register . Then 128 bytes are written (not necessarily that many - it depends on the block size), which, as in the case of a double word, are put into the buffer. As soon as the last byte is written, the entire batch is written to EEPROM/flash at once.

By the way, about **the fast writing mode** . To clear a block before quickly and decisively writing something to it, you can use the function

```
FLASH_EraseBlock(uint16_t BlockNum, FLASH_MemType_TypeDef FLASH_MemType)
```

which should also be executed from RAM. The parameters here have the same meaning as for the previous function.

For general educational purposes, you can look into RM0031 and see how block clearing is implemented:
To clear a block, you must first set the **ERASE** bit in **FLASH_CR2** , and then write zeros to the first four bytes inside this block.

It should be noted that due to an error in the MC itself, when such events as processing a DMA request and writing to EEPROM/flash coincide, the core will hang.

That's probably all. At the request of regular radio listeners, I can provide a sample code that writes something somewhere. :)

eeprom , flash , STM8L

+3 10 October 2011, 03:01 **dcoder**

Comments (5)

RSS Collapse / Expand

Add about the unpleasant moment when working with eeprom and DMA simultaneously, reflected in errate. 0

ZiB
10 October 2011, 05:55

So if we force DMA to write to eeprom, everything will freeze? And if DMA works when writing to eeprom, the same thing will happen? 0

dcoder
October 10, 2011, 10:59 PM ↑

ziblog.ru/2011/08/09/obnovili-spisok-oshibok/

0

ZiB
11 October 2011, 04:36 ↑

> *But if we need to read not a byte, but for example a word or a double word, then it is easier to do this:*

could be so:

```
uint32_t *MyEEPROMVariable = (uint32_t *)0x1000;
```

And in Cosmic there is a modifier @eeprom: www.cosmic-software.com/faq/faq13.php

0

reptile
October 10, 2011, 15:03

> *At the request of regular radio listeners, I can provide a sample code that writes something somewhere.*

That would be great. Thanks for the article.

0

megannnn98
05 December 2013, 13:44

Only registered and authorized users can leave comments.

Design by — Studio XeoArt

© Powered by LiveStreet CMS