Attention: during the operation of the debug probe (the first revision) with the target, the hardware reset of the latter is allowed exclusively by means of the debugger; any other methods of hardware reset may lead to failure of the debug probe.

Key features of the utility CMSIS-DAP[1]

- Operates with the main and informational (version 1.5 or upper) flash memory;
- Suppots mass erase;
- Saves a program from microcontroller to a file in the HEX[2] format;
- Writes a program to the microcontroller using files in the HEX[2] and ELF[3] formats;
- Compares the recorded program with the data in the files in HEX[2] and ELF[3] formats.

The software CMSIS-DAP provides:

- Downloading of the program into RAM with subsequent launch.
- Debug output via SWO interface with the ability to save the received data to a file in the TXT format.
- Reading and writing of the core registers and variables in memory.
- Hardware RESET signal generation.
- Update of the firmware of the debugger.
- The ability to execute user scripts before reading, writing and erasing of the memory.
- Support of the UART-loader by JSC «ICC Milandr» company.

[1] Download CMSIS-DAP flash programming software (the utility) at https://ic.milandr.ru/upload/iblock/665/d8vms7yinqwlkex79ddsh04ga9af8cgr/CMSIS-DAPi.zip
[2] **Requirements for the HEX format file**:
- sort addresses must be in the ascending order;
- max bytes length of a row – 16 bytes.
A file in the hex format obtained with CodeMaster ARM development environment doesn't meet these requirements, so you must follow the instruction: http://support.milandr.ru/base/spravka/32-razryadnye-mikrokontrollery/sredy-otladki-i-programmatory/36885/

[3] Only ELF format files obtained with IAR EWB and ARM KEIL environment.

Description of the program interface CMSIS-DAP

«Debug probe kit for chips with Cortex-M core TSKYA.468998.109 (the original part number is ТСКЯ.468998.109)» can work with the software «CMSIS-DAP vx.y» developed by the JSC «ICC Milandr». The general view of the program is given in Figure 1.

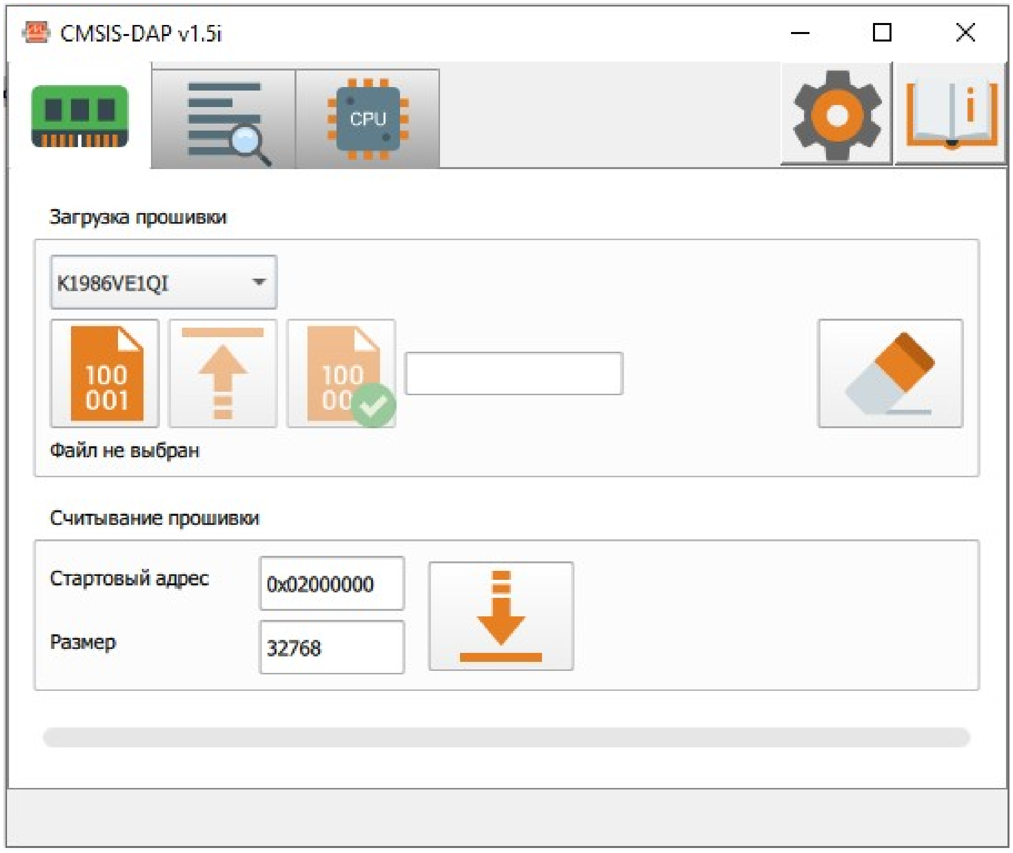


Figure 1 - The general view of the utility dialog window[4]

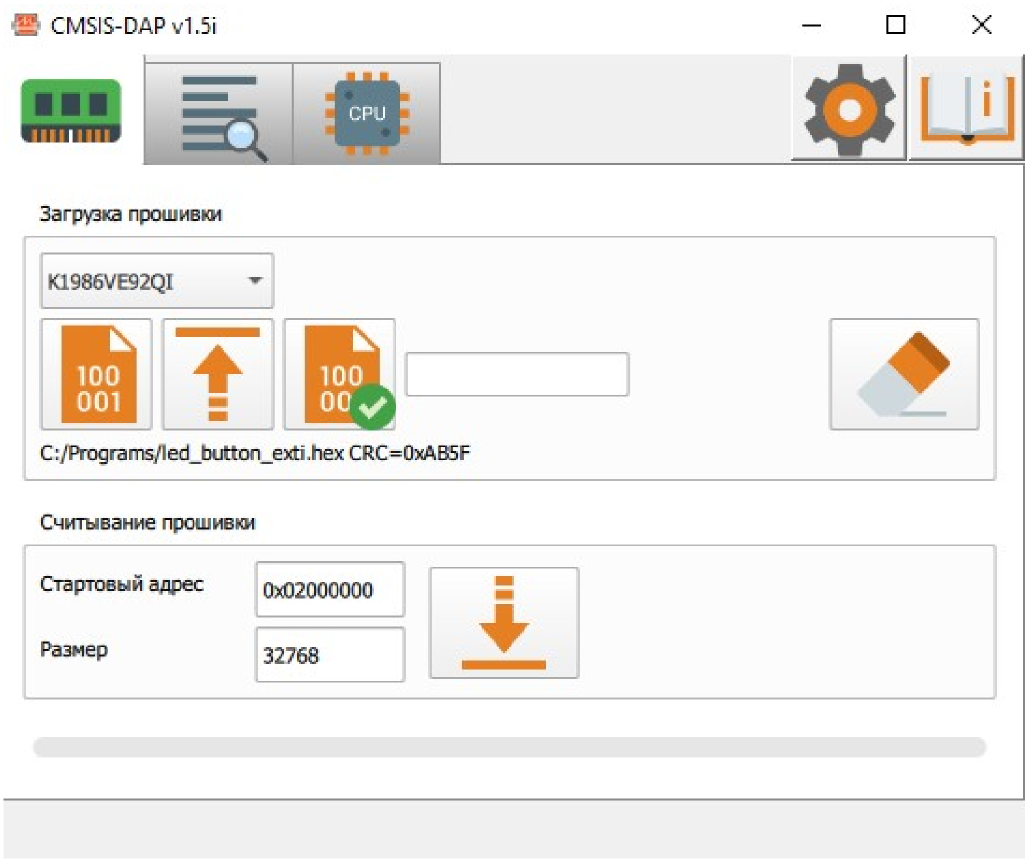
The utility contains three dialog tabs:

1. Memory programing tab;
2. SWO output data display tab;
3. Core registers and variables in memory access tab.

There are settings and help buttons in the right upper corner.

[4] The view of the main window of the «CMSIS-DAP utility» can differ depending on the utility version

Memory Programing Tab



Загрузка прошивки

K1986VE92QI

C:/Programs/led_button_exti.hex CRC=0xAB5F

Считывание прошивки

Стартовый адрес 0x02000000

Размер 32768

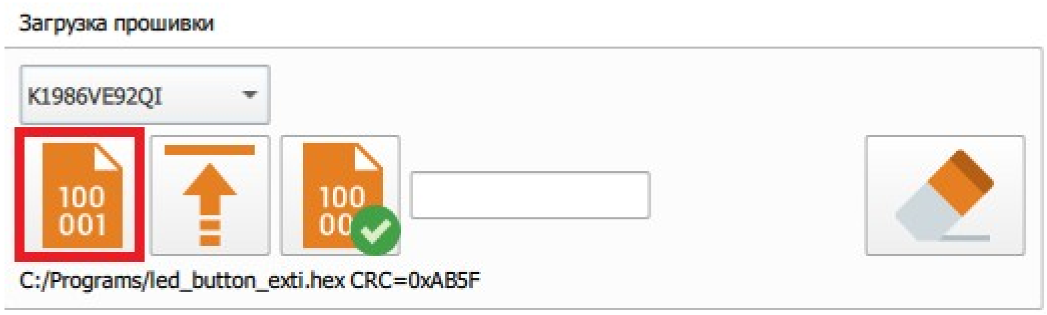
Figure 2 - Memory programing tab

The first tab of the utility «CMSIS-DAP vx.y» - «Memory programing tab» (Figure 2) allows to work with the memory of the target. Allowed operations are read, write, verification of the flash content and erase. The memory erase operation executes the mass erase of the flash of the target device.

Attention! The operation with the flash is disabled while SWO data display is running

Writing to the Memory

To download the data you need to:

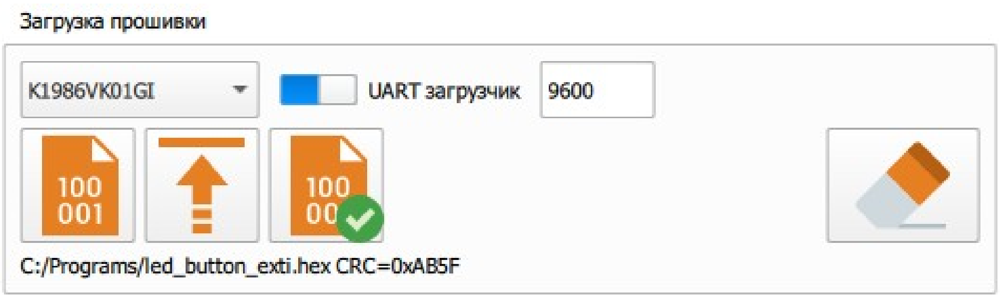


C:/Programs/led_button_exti.hex CRC=0xAB5F

Figure 3 - Selecting a file for programing

1. Select the required loader from the drop-down list;
2. Select a file (hex[2] or elf[3]) for programming. After selecting, there will be the full path to the file and its CRC value below the button «100 001»;
3. Press the button «the up arrow with the horizontal line» to start downloading and wait for writing operation to complete.

❖ If the loader supports the ability to work via UART, then you can enable this mode by switching the slider. After that, you can set the required baud rate (Figure 4).



C:/Programs/led_button_exti.hex CRC=0xAB5F

Figure 4 - UART loader set up

Then you must set the UART loader mode for the microcontroller and fulfill the connection as in Table 1.

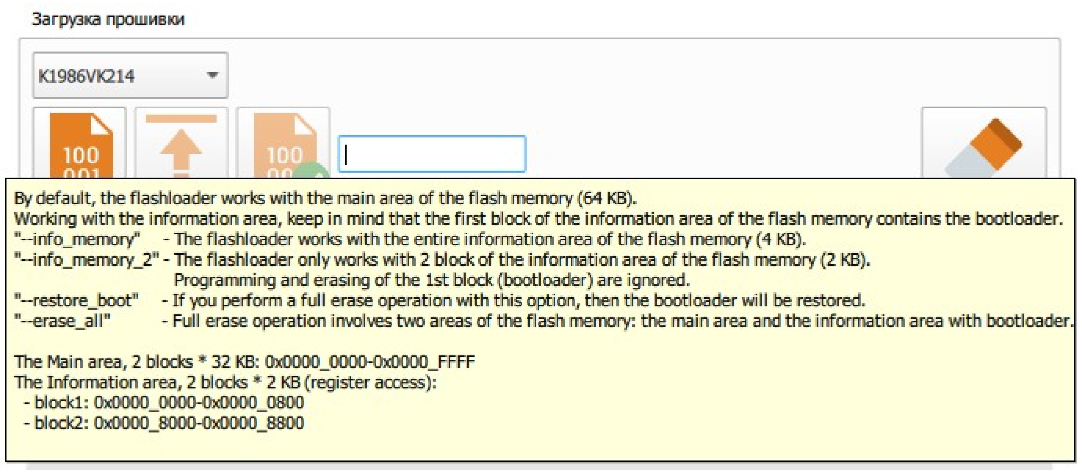
Table 1

The number of the pin of the debug probe	**The number of the pin of the target**
17	TX
19	RX
15	Reset

1	Vcc
1. Connect the grounds. 2. The connection of the Reset pin is not obligatory but if not connected you must reset the target yourself before using the UART interface[5].	

In the current version of the software the data can be written to RAM at the program launch and read into a file using the UART loader.

- If the loader supports supplementary options, then you can enter them to the field next to the button «100 00 and the tick» (the comparison data):



```
By default, the flashloader works with the main area of the flash memory (64 KB).
Working with the information area, keep in mind that the first block of the information area of the flash memory contains the bootloader.
"--info_memory"   - The flashloader works with the entire information area of the flash memory (4 KB).
"--info_memory_2" - The flashloader only works with 2 block of the information area of the flash memory (2 KB).
                    Programming and erasing of the 1st block (bootloader) are ignored.
"--restore_boot"  - If you perform a full erase operation with this option, then the bootloader will be restored.
"--erase_all"     - Full erase operation involves two areas of the flash memory: the main area and the information area with bootloader.

The Main area, 2 blocks * 32 KB: 0x0000_0000-0x0000_FFFF
The Information area, 2 blocks * 2 KB (register access):
 - block1: 0x0000_0000-0x0000_0800
 - block2: 0x0000_8000-0x0000_8800
```

Figure 5 - Popup window with additional options

When you hover over this field, a tooltip describing the available options will appear. The list of the options for each built-in loader is given in the section «Built-in loaders».

Verification of the recorded data

To verify the data you need to:

1. Select a file (hex[2] or elf[3]) for programming. After selecting the file, its full path and the CRC value will appear below the button;
2. Press the button «100 00 with the tick» to start verification and wait for the verification operation to complete;
3. The result will be displayed below the current progress status bar. The CRC of the memory data will be displayed too. (Figure 6).

[5] Attention: during the operation of the debug probe (first revision) with the microcontroller, the hardware reset of the latter is allowed to be performed exclusively by means of the debug probe; any other methods of hardware reset may lead to failure of the debug probe.

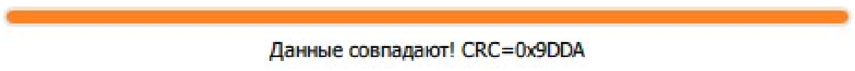
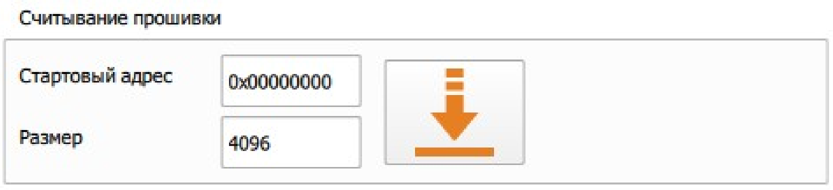

Данные совпадают! CRC=0x9DDA

Figure 6 - CRC of the memory data

Reading into a File

To read data from the memory to a file you need to:


Считывание прошивки

Стартовый адрес 0x00000000

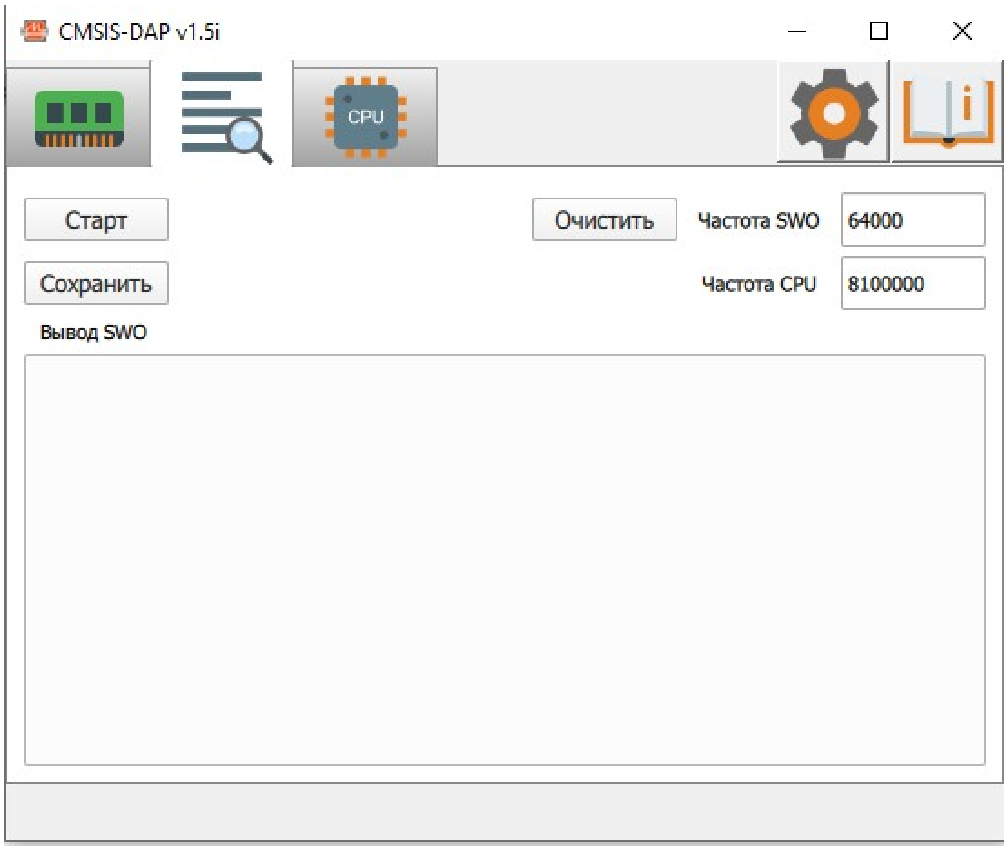
Размер 4096

Figure 7 - Reading data into a file

1. Specify the start address and the size of the loaded data in bytes;
2. Press the button «the down arrow and the horizontal line» to read into a file and specify a file name to save the data;
3. Wait for the process to complete. After that, the amount of data which has been read and CRC value will be displayed.

SWO Output Data Tab

The 'SWO Output Data' tab (Figure 8) shows the data received via the debugger using the SWO protocol. All received data can be exported to a plain text file (.txt).



Частота SWO: 64000
Частота CPU: 8100000

Figure 8 - SWO output data tab

To receive the data you need to:

1. Specify the desired SWO interface exchange frequency (lower then 1000000) (the field «Частота SWO») and set the current working frequency of the target (the field «Частота CPU»);

2. Press the button «Старт» (which means «to start»);

You need to press the button «Сохранить» (which means «to save») to save the received data to a plain text file. The button «Очистить» (which means «to clear») erases all data in the SWO output field.

Attention! SWO is unavailable if the JTAG interface has been selected or any operation with the memory is being fulfilled.

Core Registers and Memory Variables Access Tab

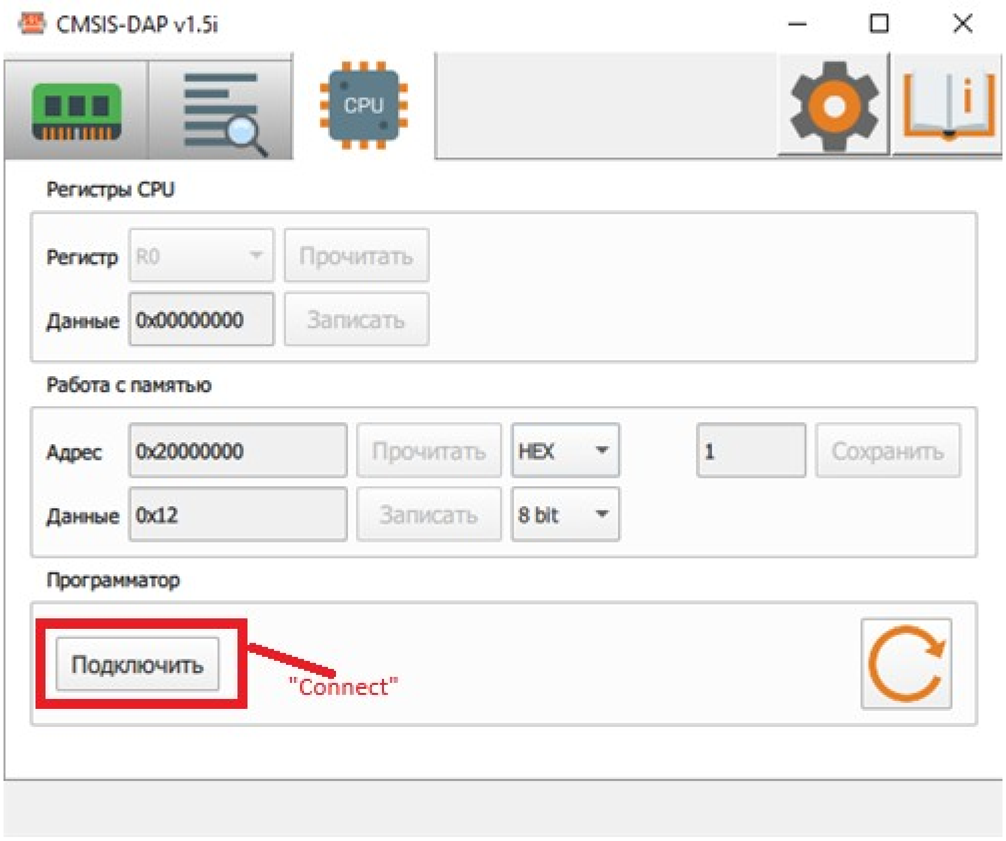


Figure 9 - Core registers and memory variables access tab

The third dialog tab, labeled «Core registers and memory variables access tab» (Figure 9), enables reading and writing of the core registers and memory variables. Press the button «Подключить» («Connect»), to enable the data entry fields. When accessing memory, you can select the variable display format: binary, hexadecimal, or decimal.

While reading core registers, the core briefly stops for about 5 ms. It’s important if the program controls the equipment which is critical to the response time.

When the reset button is pressed, a hardware RESET signal is generated (the RESET pin is set to a low level for ~100 ms).

The Utility Settings

Click the «settings» button to open the utility settings dialogue window (**Ошибка! Источник ссылки не найден.**):

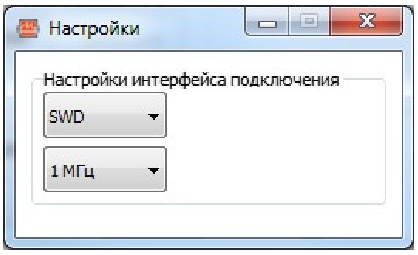
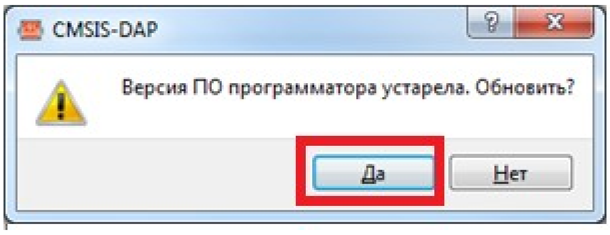


Figure 10 - The dialog window "Настройки" ("Settings")

JTAG and SWD interfaces can be enabled in this dialogue. The interface speed can be defined – 100kHz, 500kHz or 1MHz. The settings are applied immediately after closing the window.

The Debugger Firmware Update

If the firmware version of the debug probe isn't actual, the window will appear with a suggestion to update the software (Figure 11).


CMSIS-DAP

Версия ПО программатора устарела. Обновить?

Figure 11 – The notification about the necessity to update the firmware

After pressing the button «Да» («Yes») the firmware update procedure will start in the debug probe. The LED will change its color to the blue. When the update is complete, the LED will turn red again.

Pin assignment of the Debug Probe

Different variants of pin assignment of the debug probe in different working modes are shown in Figure 1:

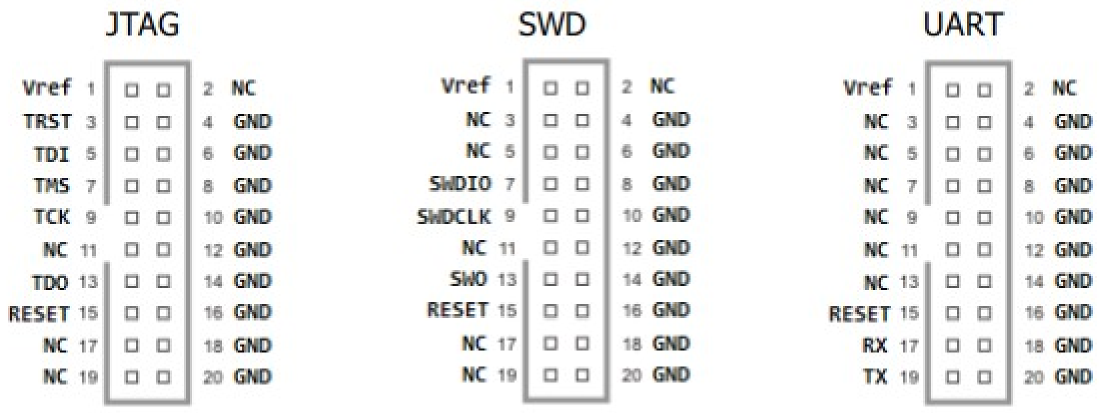


Figure 1 – Variants of pin assignment

It is necessary to provide the voltage on the Vref pin. The permissible voltage range is (2.7 - 5.5) V.

Attention! It is impossible to supply the voltage for the target board from this debugger

Description of the LED Indication of the Debugger

There is a full-color LED on the top of the case. It can turn different indications depending on its current state (Table 2).

Table 2 - Description of the LED indication of the debugger

LED state	Description
Flashing red	No USB connection
Permanent red	The debugger is in the standby mode
Permanent blue	The debugger is in the software update mode
Single blue blink	The reset pin was set to low, and then to high level
Permanent green	The debugger is connected to the target.
Blinking green	The debugger transmits data in the UART mode
Blinking red	The debugger receives data in the UART mode

If the debugger is used with the development environment then the state of the LED is defined by the IDE. For example, when working in the IAR EWB the permanent yellow signals that the application is launched while the permanent green indicates that the program is suspended.

Description of the CMSIS Utility

Working with memory

The utility uses the loader mechanism to write data to the target. The loader is a compiled program for the target. It contains program functions which provide erasing, writing and verification of the data. This program is loaded to the RAM of the target.

The program is compatible with bootloaders used in the IAR EWB. If there is the need to add one's own bootloader, one has to create it in accordance with the guide http://supp.iar.com/filespublic/updinfo/004916/arm/doc/flashloaderguide.pdf. Then the loader is compiled and 3 files are added to the folder «flashloaders» located in DAP.exe program directory:

1. loader.flash
2. loader.out
3. script.js

***.flash**

The file contains the flash memory description. Tags «macro» and «aggregate» are not used. Special tags are added:

- uart – if set to 1, then there is a UART-loader in the target device. The UART-loader by JSC «ICC Milandr» is only supported;
- uart_baud – the UART speed to synchronize with the loader;
- js – JS-code file.

***.out**

An ELF file with the loader for the flash.

***.js**

An optional file with the JS code. It is performed before reading, writing and erasing of the memory. One can reset a specific protection, stop Watchdog and do other preparatory actions in it.

When the program is launched, it scans the «flashloaders» directory for the *.flash files. When the file is found, the program attempts to open *.out file defined in the exe tag. The full path to the file is ignored, the file name is only used. When the file is successfully opened, an item with the same name as the name of the *.flash file is added to the list of the loaders.

The mass erase of the memory during the memory initialization is supported. The FLAG_ERASE_ONLY flag is set while executing FlashInitEntry to provide the mass erase. If this option is not supported by the loader, then the memory will be erased sector by sector.

The function FLASHChecksumEntry is called to check the CRC of the written data. If this function is not available in the loader, then the written data will be read with subsequent CRC calculation.

Built-in Loaders

- K1986VE92QI

Additional options

The loader works with the main flash memory (128 KB) by default (when the field is empty).

"--info_memory" – the loader works with the informational flash memory (4 KB).

"--erase_all" – mass erase operation, affects the two areas of the flash memory: the main and the informational.

- K1901VC1QI (SWD mode only)

Additional options

The loader works with the main flash memory (128 KB) by default (when the field is empty).

"--info_memory" - the loader works with the informational flash memory (4 KB).

"--erase_all" - mass erase operation, affects the two areas of the flash memory: the main and the informational.

- K1986VE1QI

Additional options

The loader works with the main flash memory (128 KB) by default (when the field is empty).

"--info_memory" – the loader works with the informational flash memory (4 KB).

"--erase_all" – mass erase operation, affects the two areas of the flash memory: the main and the informational.

- K1986VK01GI

Additional options are not supported.

- K1986VK214

Additional options

The loader works with the main FLASH-memory (64 KB) by default (when the field is empty).

Attention! While working with the informational memory it must be taken in account that the first block of informational flash memory contains the bootloader.

"--info_memory" – the loader works with the informational flash memory (4 KB).

"--info_memory_2" – the loader works only with the second block of the informational flash memory (2 KB). Programming and erasing of the first block will be ignored.

"--restore_boot" – the bootloader will be restored if mass erase is performed with this option.

"--erase_all" - mass erase operation, affects the two areas of flash memory: the main and the informational with the bootloader.

Main area, two blocks * 32 KB: 0x0000_0000-0x0000_FFFF

Informational area, two blocks * 2 KB (register access):
- block 1: 0x0000_0000-0x0000_0800
- block 2: 0x0000_8000-0x0000_8800

- K1986VK234.

The loader works with the main FLASH-memory (128 KB) by default (when the field is empty).

Attention! While working with the informational memory it must be taken in account that the first block of the informational FLASH memory contains the bootloader
"--info_memory" – the loader works with the full informational FLASH memory (8 KB).

"--info_memory_2" – the loader works only with the second block of the information flash memory (6 KB). Programming and erasing of the first block will be ignored.

"--restore_boot" – bootloader will be restored if mass erase is performed with this option.

"--erase_all" - mass erase operation, affects the two areas of the flash memory: the main and the informational with the bootloader

Main area, four blocks * 32 KB: 0x0000_0000-0x0001_FFFF

Informational area, four blocks * 4 КБ (register access):
- block 1: 0x0000_0000-0x0000_0800
- block 2: 0x0000_8000-0x0000_8800
- block 3: 0x0001_0000-0x0001_0800
- block 4: 0x0001_8000-0x0001_8800

CRC Calculation Algorithm

When opening a file, verifying and reading the data, the CRC of data is displayed. CRC calculation algorithm is shown below:

```
uint16_t calcCrc16( uint8_t *data, uint32_t size, uint16_t crc )
{
    while (size--)
    {
        int i;
        uint8_t byte = *data++;

        for (i = 0; i < 8; ++i)
        {
            uint32_t osum = crc;
            crc <<= 1;
            if (byte & 0x80)
                crc |= 1 ;
            if (osum & 0x8000)
                crc ^= 0x1021;
            byte <<= 1;
        }
    }
    return crc;
}
```

The data CRC is calculated as presented below to provide the compatibility with the Crc16 function from the bootloader:

```
    uint8_t zero[2] = { 0, 0 };
    uint16_t crc = 0;

    crc = _calcCrc16( someData, someDataSize, crc );
    crc = _calcCrc16( zero, 2, crc );
```

Working with Script Files

There is a possibility to perform the JS-code from the file before writing, erasing or reading of the memory. You need to add tag to a *.flash file:

```
<flash_device>
  …
  <js>script.js</js>
  …
</flash_device>
```

The global variable «event» was added to provide the ability to determine the current event. An example which demonstrates possible states of the «event» variable is shown below:

```
if( event == "save" )
  {
      // script is called before read to file procedure
  }
  else if( event == "program" )
  {
      // script is called before write to memory procedure
  }
  else if( event == "erase" )
  {
      // script is called before erase procedure
  }
```

Currently, the following functions are available:

- `void dap.showMessage("Text")` – shows message in program status field;
- `unsigned int dap.readMemory32( unsigned int adr )` – reads memory at a given address;
- `bool dap.writeMemory32( unsigned int adr, unsigned int val )` – writes data to the memory at a given address. Returns «true» if writing was successful;
- `unsigned int dap.readDpReg( unsigned int reg )` – reads the register Debug port;
- `bool dap.writeDpReg( unsigned int reg, unsigned int val )` – writes to the Debug port register. Returns «true» if writing was successful;
- `unsigned int dap.readApReg( unsigned int reg )` – reads the Access port register;
- `bool dap.writeApReg( unsigned int reg, unsigned int val )` – writes to the Access port register. Returns «true» if writing was successful.

The example of the script for the microcontroller NRF52XX is shown below:

```
var reg, result;

dap.writeDpReg( 0x08, 0x01000000 );      // Switch to second AP (0x08 - DP_SELECT)
reg = dap.readApReg( 0x0C );             // 0x0C (APPROTECTSTATUS) Protection status AP

if( reg === 0 )                          //  0 = protection is enabled
{
            dap.showMessage("Disable access port protection: start");
            dap.writeApReg( 0x04, 0x00000001 );     // 0x04 (ERASEALL) – erase full FLASH

   while( dap.readApReg( 0x08 ) === 1 ){};          // 0x08 (ERASEALLSTATUS) – FLASH erase status
            dap.showMessage("Disable access port protection: done");
}
else
            dap.showMessage("No protection enabled");

dap.writeDpReg( 0x08, 0x00000000 );      // switch to first AP (0x08 - DP_SELECT)

result = "Success";
```

Problems Solving

If there is an error notification or a failure of the utility, a log, containing the diagnostic information, can be created – launch the program with the «-d» option. A logFile.txt will be created in the program directory.

If you find an error in the manual or a problem in the software, please report it to us (support@milandr.ru) and we will try to assist you as soon as possible.

Manual versions

№	Date	Version	Summary	Description
1.	30.03.2022	1.0	Document was created	-
2.	14.07.2022	1.1	**Changed**: Figures **Added:** the links to the figures and tables; the descriptions of the loaders	Follow the text, 13-15