

Внимание: в процессе работы программатора (первой ревизии) с микроконтроллером, аппаратный сброс последнего допускается производить исключительно средствами программатора, любые другие способы аппаратного сброса могут привести к выходу из строя программатора.

Основные возможности программы

Работа с FLASH памятью:

- полное стирание памяти;
- сохранение программы из микроконтроллера в файл формата HEX¹;
- запись программы в микроконтроллер из файлов форматов HEX¹ и ELF²;
- сверка записанной программы с данными из файлов форматов HEX¹ и ELF².

Загрузка программы в ОЗУ с последующим запуском.

Отладочный вывод с помощью интерфейса SWO с возможностью сохранения принятых данных в файл формата TXT.

Чтение и запись регистров ядра, переменных в памяти.

Формирование аппаратного сигнала RESET.

Обновление встроенного программного обеспечения программатора.

Возможность исполнения пользовательских скриптов перед чтением, записью и стиранием памяти.

Поддержка UART-загрузчика фирмы АО «ПКК Миландр».

¹ **Используемый hex-файл должен удовлетворять требованиям:**

- сортировка адресов в порядке возрастания;
- ограничение максимальной длины байт в строке – 16 байт.

Нех-файл, полученный в среде CodeMaster ARM, не удовлетворяет этим условиям, поэтому необходимо воспользоваться инструкцией <http://support.milandr.ru/base/spravka/32-razryadnye-mikrokontrollery/sredy-otladki-i-programmatory/36885/>

² Только elf-файлы, полученные в средах WB IAR и ARM KEIL

Описание интерфейса программы CMSIS-DAP

При работе с «Комплектом программатора для микросхем с ядром CORTEX-M ТСКЯ.468998.109» возможно применить программное обеспечение, разработанное компанией АО «ПКК Миландр». На рисунке (Рисунок 1) приведён общий вид программы:

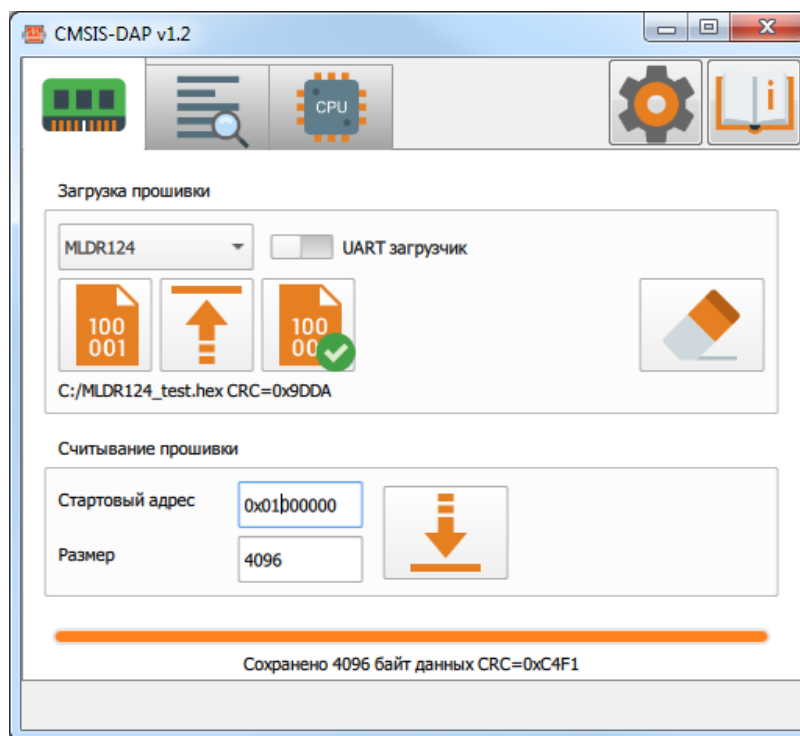


Рисунок 1 - Общий вид программы³

Программа содержит три вкладки:

1. вкладка программирования памяти;
2. вкладка отображения вывода данных SWO;
3. вкладка доступа к регистрам ядра и переменным в памяти.

Также в верхнем правом углу расположены кнопки вызова настроек и справки.

³ Вид управляющего окна утилиты может отличаться в зависимости от версии утилиты

Вкладка программирования памяти

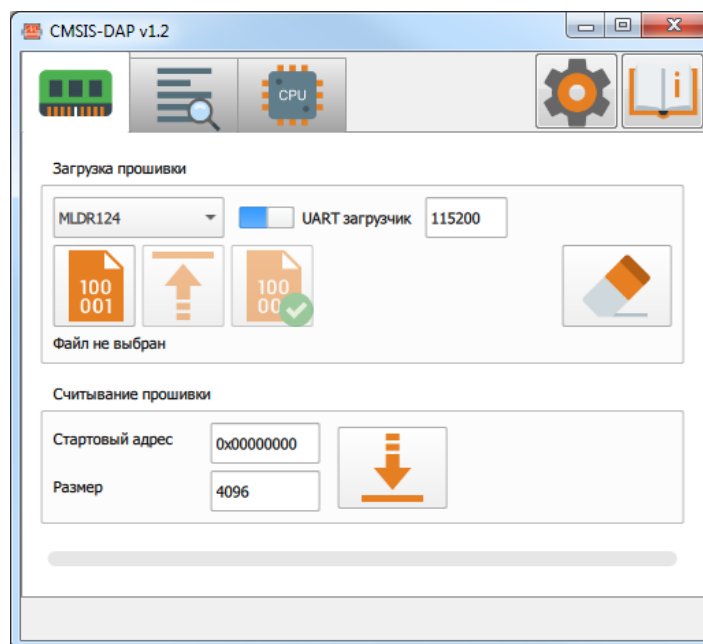


Рисунок 2 - Вкладка программирования памяти

На первой вкладке программы возможно осуществить работу с памятью микроконтроллера. Доступны операции записи, чтения, сверки и стирания памяти. Операция стирания выполняет полное стирание FLASH-памяти микроконтроллера.

Работа с памятью недоступна, пока запущено отображение данных SWO!

Запись данных в память

Для загрузки данных необходимо:

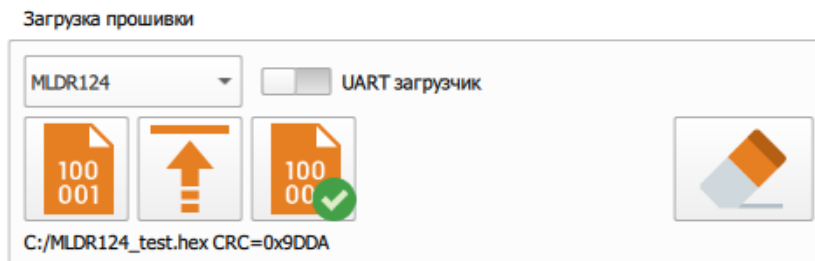


Рисунок 3

1. выбрать из выпадающего списка необходимый загрузчик. Загрузчик 1986BE9x позволяет работать с микроконтроллерами 1986BE91, 1986BE92, K1986BE92QI, 1986BE93, 1986BE94;
2. выбрать файл прошивки формата hex¹ или elf². После выбора файла под кнопкой отобразится полный путь до файла и значение его CRC;
3. нажать на кнопку записи прошивки и дождаться завершения процедуры записи.

- ❖ Если загрузчик поддерживает возможность работы через UART, то можно задействовать данный режим, переключив ползунок. После этого можно задать требуемую скорость соединения:

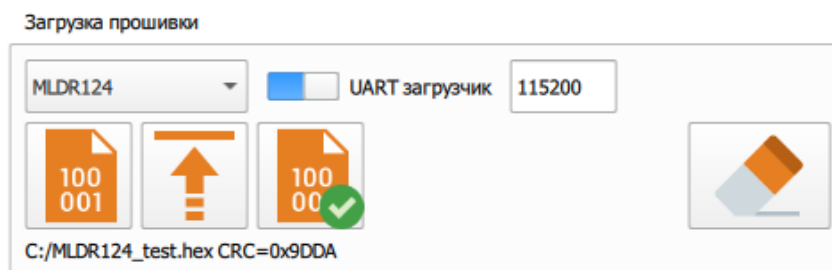


Рисунок 4

Далее необходимо выставить на микроконтроллере режим загрузки по UART. Потом подключить 17 вывод программатора к выводу TX микросхемы, 19 вывод программатора - к выводу RX, 15 вывод программатора - к выводу RESET, 1 вывод программатора - к выводу VCC, соединить «земли» (подключение вывода RESET не обязательно, но тогда

перед работой по интерфейсу UART необходимо самостоятельно осуществить сброс микросхемы⁴.

На текущий момент с помощью загрузчика UART возможно записывать данные в ОЗУ с запуском программы на выполнение и считывать данные в файл.

- ❖ Если загрузчик поддерживает дополнительные опции, то их можно ввести в поле рядом с кнопкой сверки данных:

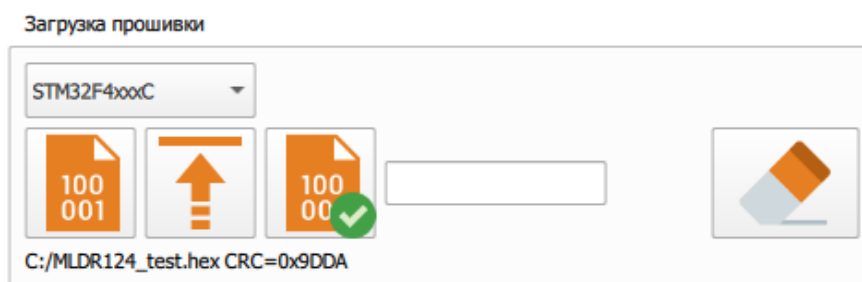


Рисунок 5

При наведении курсора на данное поле появится подсказка с описанием доступных параметров.

Сверка записанных данных

Для сверки данных необходимо:

1. выбрать файл прошивки формата hex¹ или elf². После выбора файла под кнопкой отобразится полный путь до файла и значение его CRC;
2. нажать на кнопку запуска процедуры сверки данных и дождаться завершения процесса;
3. результат будет отображен под шкалой текущего прогресса выполнения. Также будет отображено CRC данных из памяти:

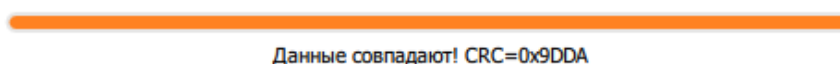


Рисунок 6 - CRC данных из памяти

⁴ **Внимание:** в процессе работы программатора (первой ревизии) с микроконтроллером, аппаратный сброс последнего допускается производить исключительно средствами программатора, любые другие способы аппаратного сброса могут привести к выходу из строя программатора.

Считывание данных в файл

Для считывания данных из памяти в файл необходимо:

Считывание прошивки


Стартовый адрес	<input type="text" value="0x00000000"/>	
Размер	<input type="text" value="4096"/>	

Рисунок 7 - Считывание данных в файл

1. указать стартовый адрес и размер загружаемых данных в байтах;
2. нажать на кнопку считывания данных в файл и задать имя файла для сохранения;
3. дождаться завершения операции. По завершении будет отображено количество прочитанных данных и значение их CRC.

Вкладка отображения вывода данных SWO

На второй вкладке в окне программы (Рисунок 8) отображаются данные, поступающие через отладочный вывод с помощью интерфейса SWO, с возможностью сохранения принятых данных в файл формата TXT.

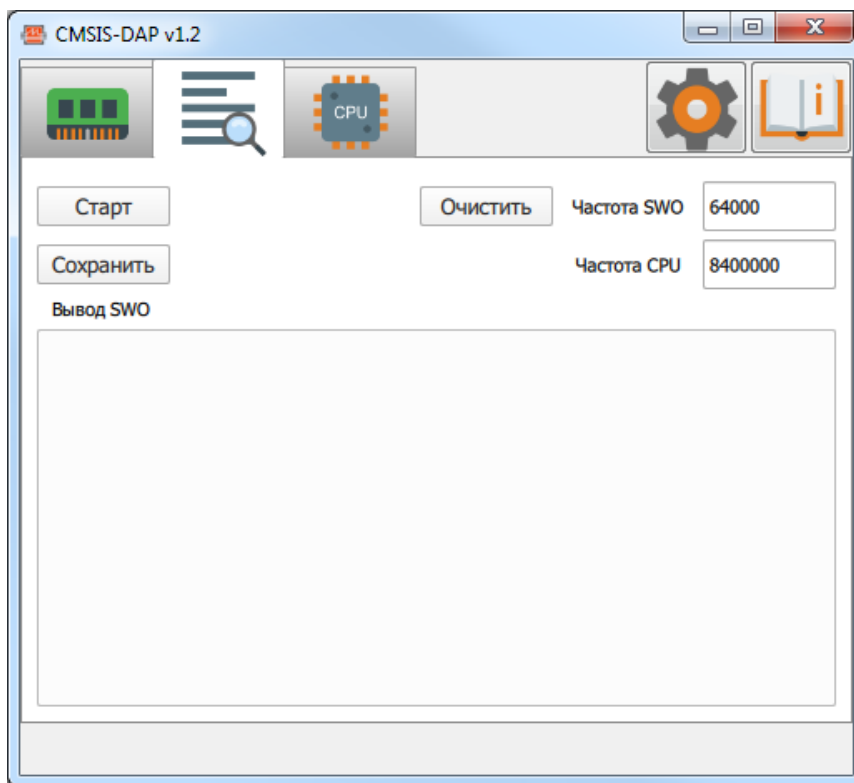


Рисунок 8 - Вкладка отображения вывода данных SWO

Для получения данных необходимо:

1. задать желаемую частоту обмена по интерфейсу SWO (не более 1000000), и указать текущую частоту работы микросхемы;
2. нажать на кнопку «Старт»;

Для сохранения полученных данных в текстовый файл необходимо нажать кнопку «Сохранить». Кнопка «Очистить» очищает поле вывода данных SWO.

SWO недоступно, если выбран интерфейс подключения JTAG или выполняются операции работы с памятью!

Вкладка доступа к регистрам ядра и переменным в памяти

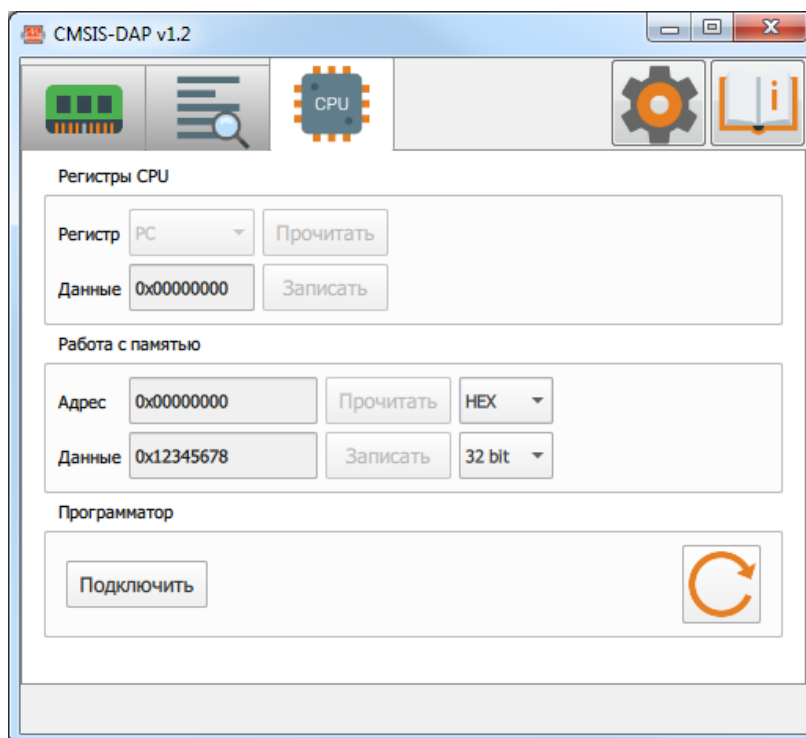


Рисунок 9 - Вкладка доступа к регистрам ядра и переменным в памяти

На третьей вкладке в окне программы (Рисунок 9) осуществляется чтение и запись регистров ядра, переменных в памяти. Для этого нужно нажать кнопку «Подключить», после чего станут активными поля ввода данных. При работе с памятью можно выбрать в каком виде отображать значение переменной в памяти (двоичное, шестнадцатеричное или десятичное).

При чтении регистров ядра происходит его кратковременная остановка на время порядка 5 мс. Нужно учитывать это, если программа выполняет управление оборудованием, критичному к времени отклика.

При нажатии на кнопку сброса выполняется формирование аппаратного сигнала RESET (вывод RESET переводится в низкий уровень на ~100 мс).

Настройки программы

При нажатии на кнопку настроек появится окно с настройками программы:

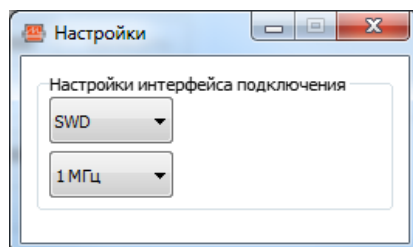


Рисунок 10 - Окно "Настройки"

Поддерживаются интерфейсы подключения JTAG и SWD на скоростях 100 кГц, 500 кГц и 1 МГц. Настройки применяются сразу после закрытия окна.

Обновление ПО программатора

Если ПО в программаторе не соответствует актуальной версии, появится окно с предложением об обновлении:

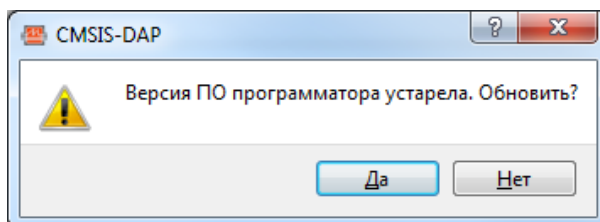


Рисунок 11 - Предупреждение о необходимости обновить версию программы

После нажатия на кнопку «Да» начнётся процедура обновления ПО в программаторе. Светодиод изменит цвет свечения на синий. По завершении обновления светодиод снова станет красным.

Назначение выводов разъема программатора

Назначение выводов разъема программатора в различных режимах работы приведено на рисунке:

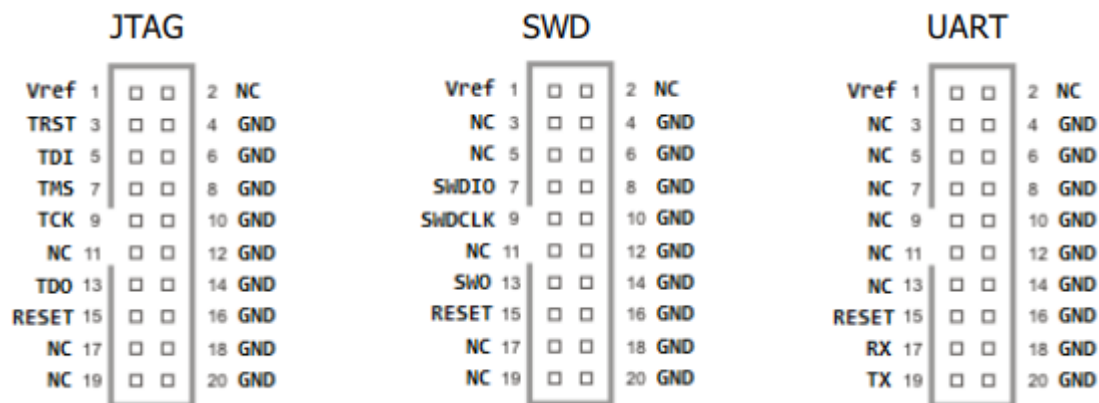


Рисунок 12 - Назначение выводов разъема программатора

Для работы программатора необходимо наличие напряжения на выводе Vref. Допустимый диапазон напряжений (2,7 - 5,5) В.

Подать питание на целевую плату с данного программатора невозможно!

Описание световой индикации программатора

На верхней части корпуса расположен полноцветный светодиод. В зависимости от текущего состояния он может принимать различные состояния:

Таблица 1 - Описание световой индикации программатора

Состояние светодиода	Описание
Мигает красный	Нет подключения по USB
Горит красный	Программатор в режиме ожидания
Горит синий	Программатор в режиме обновления прошивки
Однократно моргнул синий	Вывод RESET был переведён в низкий, а потом в высокий уровень
Горит зелёный	Программатор подключен к микроконтроллеру
Моргает зелёный	Программатор в режиме UART передаёт данные
Моргает красный	Программатор в режиме UART принимает данные

Если программатор используется в связке со средой разработки, то состояние светодиода определяется этой средой. Например, в среде разработки IAR постоянно горящий желтый говорит о том, что программа сейчас выполняется, а зелёный – программа приостановлена.

Описание функционала программы

Работа с памятью

Для записи данных в FLASH-память микроконтроллера программа использует механизм загрузчиков. Загрузчик представляет из себя скомпилированную под целевой микроконтроллер программу с функциями стирания, записи и проверки данных. Данная программа загружается в ОЗУ микроконтроллера.

Программа совместима с загрузчиками, используемыми в IDE IAR. Для добавления своего загрузчика необходимо создать загрузчик в соответствии с документом <http://supp.iar.com/filespublic/updinfo/004916/arm/doc/flashloaderguide.pdf>. Далее загрузчик компилируется, и в папку flashloaders, расположенной в директории программы DAP.exe, добавляются 3 файла:

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

***.flash**

В данном файле описывается FLASH-память. Теги macro и aggregate в программе не используются. Введены дополнительные теги:

- uart – если стоит 1, то есть UART-загрузчик в данном микроконтроллере. Поддерживается только UART-загрузчик фирмы АО «ПКК Миландр»;
- uart_baud – скорость UART, на которой производится синхронизация с загрузчиком;
- js – задание файла с JS-кодом.

***.out**

ELF-файл с программой загрузчика FLASH-памяти.

***.js**

Опциональный файл с JS-кодом. Выполняется перед чтением, записью и стиранием памяти. В нём можно сбросить специфичную защиту, остановить Watchdog и сделать прочие подготовительные действия.

При запуске программы происходит сканирование директории «flashloaders» на наличие файлов *.flash. При нахождении данного файла происходит попытка открытия файла *.out, указанного в тэге exe. Полный путь файла игнорируется, используется только его имя. При успешном открытии файла в список загрузчиков добавляется пункт с именем, как у файла *.flash.

Поддерживается возможность полного стирания при инициализации памяти. Для этого выставляется флаг **FLAG_ERASE_ONLY** при выполнении FlashInitEntry. Если загрузчик не поддерживает такой возможности, то будет выполнено посекторное стирание всей памяти.

Для проверки CRC записанных данных вызывается функция FlashChecksumEntry. Если данная функция не реализована в загрузчике, то происходит вычитывание записанных данных назад с последующим вычислением CRC.

Алгоритм вычисления CRC

При открытии файла с данными, сверке и считывании данных отображается CRC данных. Ниже отображен алгоритм вычисления CRC:

```
uint16_t calcCrc16( uint8_t *data, uint32_t size, uint16_t crc )
{
    while (size-->0)
    {
        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;
}
```

Для совместимости с функцией Crc16 из загрузчика, CRC данных считается следующим образом:

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

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

Работа со скриптами

В программе реализована возможность выполнять JS-код из файла перед записью, стиранием и чтением памяти. Для этого необходимо в файл *.flash добавить тэг:

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

Для возможности определения текущего события добавлена глобальная переменная event. Ниже пример с её возможными состояниями:

```
if( event == "save" )
{
    // скрипт вызван перед процедурой чтения памяти в файл
}
else if( event == "program" )
{
    // скрипт вызван перед процедурой записи в память
}
else if( event == "erase" )
{
    // скрипт вызван перед процедурой стирания всей памяти
}
```

На текущий момент доступны следующие функции:

- `void dap.showMessage("Text")` – отображение сообщения в статусной строке программы;
- `unsigned int dap.readMemory32(unsigned int adr)` – чтение памяти по заданному адресу;
- `bool dap.writeMemory32(unsigned int adr, unsigned int val)` – запись памяти по заданному адресу. Возвращает true при успешной записи;
- `unsigned int dap.readDpReg(unsigned int reg)` – чтение регистра Debug port;
- `bool dap.writeDpReg(unsigned int reg, unsigned int val)` – запись регистра Debug port. Возвращает true при успешной записи;
- `unsigned int dap.readApReg(unsigned int reg)` – чтение регистра Access port;
- `bool dap.writeApReg(unsigned int reg, unsigned int val)` – запись регистра Access port. Возвращает true при успешной записи.

Ниже приведён пример скрипта для микроконтроллера NRF52XX:

```
var reg, result;

dap.writeDpReg( 0x08, 0x01000000 ); // Переключаемся на второй AP (0x08 - DP_SELECT)
reg = dap.readApReg( 0x0C );        // 0x0C (APPROTECTSTATUS) находится статус защиты AP

if( reg === 0 )                    // 0 = защита включена
{
    dap.showMessage("Disable access port protection: start");
    dap.writeApReg( 0x04, 0x00000001 ); // 0x04 (ERASEALL) - стирание всей FLASH

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

dap.writeDpReg( 0x08, 0x00000000 ); // Переключаемся на первый AP (0x08 - DP_SELECT)

result = "Success";
```

Решение проблем

В случае возникновения сообщения об ошибке или некорректного функционирования программы существует возможность вывода диагностического файла при работе программы. Для этого необходимо запустить программу с ключом –d. В результате в директории программы будет создан файл logFile.txt

Версия 2.5

Дата редактирования файла: 06/11/2020