Содержание

Утилита SAUFLASH

Rev. 1.0 beta

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.
РУКОВОДСТВО ПО ПОРТИРОВАНИЮ.

1. Введение

1.1. Назначение утилиты.

Утилита предназначена для программирования микросхем, содержащих постоянную память (Flash, EEPROM) , подключенных к интерфейсам микропроцессоров и микроконтроллеров фирмы Texas Instruments. Также данная утилита может быть расширена до возможности загрузки данных в ОЗУ (например, конфигурирование FPGA, подключенной своим загрузочным интерфейсом к процессору), тестирования систем (тест ОЗУ, подсчет контрольной суммы ПЗУ, другие тестовые функции) и решения других задач, требующих исполнения кода в процессоре под контролем отладочных средств и передачи данных между компьютером устройством по интерфейсу JTAG.

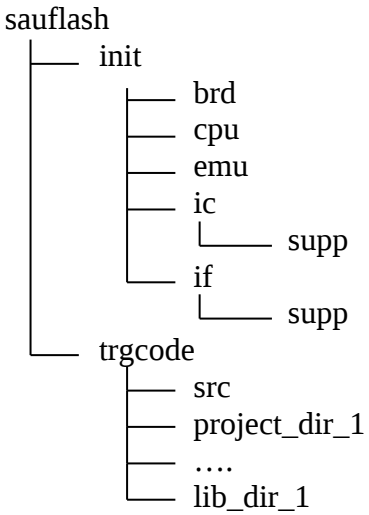
2. Принцип работы утилиты.

2.1. Состав утилиты

Утилита SAUFLASH состоит из двух компонентов. Первый компонент представляет собой набор скриптов на языке TCL (Tool Command Language) с использованием библиотеки-расширения TCLXDS (© Sauris GmbH), обеспечивающих обмен данными и командами с устройством на базе процессора фирмы TI с использованием отладочных средств этого процессора. Данный компонент исполняется на PC. Второй компонент представляет собой программу для процессора устройства, исполняемую под контролем первого компонента, и производящую непосредственные операции с программируемой микросхемой памяти.

2.2. Структура и расположение компонентов.

В процессе инсталляции утилиты создается следующая структура директорий на диске:



Основной скрипт расположен в директории **sauflash**. В директории **trgcode** находятся исполняемые модули для различных процессоров, которые собственно программируют микросхемы памяти (или on-chip память). В ее поддиректории **src** находятся исходные тексты этих исполняемых модулей. Все исходные тексты представляют собой проекты для TI Code Composer Studio v3.3 SR12.1, за исключением модуля для TMS320VC33, который сделан в TI Code Composer 4.10 SP2. В оставшихся директориях находятся файлы проектов для разных процессоров, использующие данные исходные тексты, и библиотеки функций программирования внутренней памяти, и сопутствующие им временные файлы.
В директории **init** расположены скрипты-инициализаторы для различных компонентов устройства. В поддиректории **emu** находятся скрипты, инициализирующие параметры эмуляторов, в **brd** – параметры плат (окружения процессора), в **cpu** – собственно процессора, в **if** – интерфейса процессора, через который подключена программируемая ИМС, и в **ic** – параметры собственно микросхем памяти. В поддиректориях **supp** директорий **ic** и **if** расположены дополнительные скрипты, ссылки на которых находятся в скриптах-инициализаторах, отвечающие за физическую инициализацию процессора, связанную с конкретным интерфейсом и/или ИМС памяти.

2.3. Запуск утилиты. Формат командной строки.

Утилита имеет следующий формат командной строки:

Программирование:
sauflash <emu>:<board>:<cpu>:<iface>:<IC> program <file_name> [<address_offset>]

Стирание одного сектора:
sauflash <emu>:<board>:<cpu>:<iface>:<IC> erasesector <address_offset>

Стирание ИМС целиком:
sauflash <emu>:<board>:<cpu>:<iface>:<IC> erasechip

<emu> - эмулятор. Одно из имен из директории sauflash/init/emu без расширения.
<board> - плата. Одно из имен из директории sauflash/init/brd без расширения.
<cpu> - процессор. Одно из имен из директории sauflash/init/cpu без расширения.
<iface> - интерфейс. Одно из имен из директории sauflash/init/if без расширения.
<IC> - ИМС памяти. Одно из имен из директории sauflash/init/ic без расширения.

<file_name> - путь к файлу, содержащему данные для программирования в ИМС. Формат файла – бинарный, т.е. каждый байт файла физически программируется в каждый байт микросхемы памяти, вне зависимости от того, куда и каким образом подключена эта ИМС, и какой она разрядности. С точки зрения пользователя программирование ИМС данными из этого файла при помощи данной утилиты эквивалентно программированию данными из этого же файла отдельным программатором.

<address_offset> - для команды программирования – смещение в 8-битных байтах внутри микросхемы памяти, начиная с которого должны быть запрограммированы данные из файла. Для команды стирания сектора – адрес внутри микросхемы памяти, попадающий в сектор, подлежащий стиранию.
Внимание! *Адреса и смещения никак не связаны ни с типом процессора, ни с шириной его шины данных, ни с разрядностью ИМС. Адрес* ***всегда считается в 8-битных байтах*** *внутри ИМС памяти, рассматриваемой отдельно от всей остальной системы, даже если это внутренняя память.*

Пример 1:
sauflash sau510iso:evm5502:vc5502:mcbsp0:25f512 program myprogram.bin

при помощи данной команды содержимое файла “myprogram.bin” программируется в микросхему типа 25F512, подключенную через интерфейс MCBSP0 к процессору VC5502, расположенному на плате EVM5502, подключенной к PC через эмулятор SAU510ISO.

Пример 2:
sauflash sau510iso:my2806brd:f2806:internal:flash erasechip

данная команда стирает весь массив внутренней памяти, расположенной в процессоре TMS320F2806, который расположен на плате “MY2806BRD”, подключенной к PC через эмулятор SAU510ISO.

2.4. Взаимодействие между компонентами

Последовательность действий при работе утилиты следующая:
- По очереди считываются конфигурации эмулятора, платы, процессора, интерфейса и ИМС памяти, исполняя указанные в командной строке скрипты-инициализаторы.
- Производится соединение PC и устрйоства через эмулятор, производится сброс CPU.
- Производятся инициализация платы и процессора. Вызывается функция CPUSpecificInit (определена скриптом инициализатором процессора), затем UserInitAfterReset (определена скриптом инициализатором платы). В этих функциях производится инициализация тех частей процессора и платы, которые необходимо произвести до загрузки кода программы. Например, инициализация PLL, разблокирование CSM/MSM, инициализация карт распределения памяти, и т.п.
- В CPU загружается код программы (из **sauflash/trgcode**). Имя файла с ней конфигурируется в скрипте-инициализаторе CPU.
- При необходимости устанавливается точка останова, если она отсутствует в программе изначально.
- Программа запускается на выполнение, происходит ожидание точки останова.
- Производятся дополнительные инициализации. Скрипт из sauflash/init/if/supp (ссылка на него в скрипте-инициализаторе интерфейса), затем из sauflash/init/ic/supp (ссылка на него в скрипте-инициализаторе ИМС памяти), затем функция UserInitCPUEnv, описанная в скрипте-инициализаторе платы. Эти скрипты передают информацию об интерфейсе и ИМС программе, загруженной в процессор, инициализируют интерфейс и прочие необходимые для успешного программирования ИМС части устройства.
- при команде на программирование утилита передает программе в устройстве блок данных из файла, запускает программу на исполнение, ждет достижения точки останова, анализирует статус, возвращаемый программой, и если программирование блока успешно, то происходит переход к передаче следующего блока, до тех пор, пока все данные из файла не будут запрограммированы.
- при командах стирания утилита передает программе в устройстве команду на стирание сектора и адрес внутри него, запускает программу на исполнение, ждет достижения точки останова, анализирует статус, возвращаемый программой, и если стирание успешно, то либо переходит к стиранию следующего сектора, либо завершает работу.
- по завершении операции исполняется функция UserFinishCPUEnv, описанная в скрипте-инициализаторе платы. В этой функции можно произвести какие либо действия с устройством, необходимые после окончания работы с ИМС памяти. Например перевести какие либо порты в изначальное состояние, и т.п.

2.5. Особенности вызова утилиты.

Так как утилита сама по себе является командным файлом (**<sauris_install_dir>/bin/sauflash.cmd**), то при вызове из других командных файлов необходимо пользоваться синтаксисом “*call sauflash <parameters>*” или “*cmd /c sauflash <parameters>*”.
При инсталляции путь к утилите прописывается в системную переменную окружения “PATH”, по этому указание полного пути к утилите при ее запуске не требуется.

3. Поддержанные и протестированные процессоры, интерфейсы и микросхемы памяти.

TMS320VC33
- 32-bit DATABUS BYTE 0/1/2/3
- 32-bit DATABUS 16-bit word 0/1
  - AM29LV400BT
  - AM29LV400BB

TMS320VC5501
TMS320VC5502
- MCBSPx
  - AT25xxx
  - AT25Fxxx
  - Совместимые с ними других фирм (ST, Ramtron, Microchip, e.t.c)
- I2C
  - 24Cxxx
  - Совместимые с ними других фирм (Atmel, ST, Microchip, e.t.c)

TMS320C6416
- MCBSPx
  - AT25xxx
  - AT25Fxxx
  - Совместимые с ними других фирм (ST, Ramtron, Microchip, e.t.c)
- EMIFB byte 0/1
- EMIFB 16-bit word
- EMIFA byte 0/1/2/3
- EMIFA 16-bit word 0/1
  - AM29LV400BT
  - AM29LV400BB

TMS470R1B1
- INTERNAL FLASH
  - On-chip flash array
- SPIx
  - AT25xxx
  - AT25Fxxx
  - Совместимые с ними других фирм (ST, Ramtron, Microchip, e.t.c)
- I2Cx
  - 24Cxxx
  - Совместимые с ними других фирм (Atmel, ST, Microchip, e.t.c)

TMS320F2801/2/6/8
- INTERNAL FLASH
  - On-chip flash array
- SPIx
  - AT25xxx
  - AT25Fxxx
  - Совместимые с ними других фирм (ST, Ramtron, Microchip, e.t.c)
- I2C
  - 24Cxxx
  - Совместимые с ними других фирм (Atmel, ST, Microchip, e.t.c)

4. Портирование под работу с собственным устройством, не поддержанным в стандартной комплектации.

4.1. Скрипт-инициализатор эмулятора.

Скрипты этого назначения расположены в директории **<sauris_install_dir>/sauflash/init/brd**. В скрипте-инициализаторе эмулятора определяется собственно используемый эмулятор (командой xds add) и его параметры (командами xds param). Подробное описание этих команд Вы найдете в документе “Пакет расширения TCL XDS V1.0. Справочное руководство.”

4.2. Скрипт-инициализатор платы.

Скрипты этого назначения расположены в директории **<sauris_install_dir>/sauflash/init/brd**. В скрипте-инициализаторе параметров платы определяются параметры и функции, специфичные для платы. Первом делом при помощи команд “xds add” формируется описание JTAG-цепочки, и переменной “dev” присваивается идентификатор того процессора из цепочки, через который будет производиться программирование. Затем определяется тактовая частота процессора и периферии. Для процессоров TMS320F28xx задается переменная quartz_frequency, показывающая частоту кварцевого резонатора, подключенного к процессору, для остальных процессоров задается две переменные – cpu_clock и periph_clock, показывающие, на какой частоте в данной плате работает ядро процессора и на какой периферия. Далее могут определяться пароли разблокировки – для серий 28xx это “csm_password” (пароль разблокировки CSM), для TMS470 – msm_password и flash_password (пароль разблокировки MSM и пароль разрешения операций с flash-памятью). Затем, в случае использования на плате микросхемы EEPROM с SPI-интерфейсом определяется ряд параметров, показывающих к какому разряду какого порта процессора подключен ее сигнал “CS” (выбор кристалла). Это параметры cs_port_address/cs_port_mempage, показывающее расположение регистра порта в адресном пространстве процессора (mempage – полная аналогия PAGE в .cmd файлах редактора связей (линкера), указание типа адресного пространства – команд, данных или ввода-вывода), cs_port_width, показывающий разрядность этого регистра, cs_port_bit – маска бита управления сигналом CS, и cs_port_polarity, показывающий активный уровень сигнала CS, 0 или 1. В некоторых случаях (на сегодня это только для процессоров TMS470R1) вместо полного списка всех параметров сигнала CS допустимо указание “cs_port_address auto”, в случае если сигнал CS подключен к специально предназначенному выводу порта SPI-контроллера, к которому подключена микросхема. Затем определяется функция UserInitCPUEnv, в которой производятся все необходимые для корректной работы платы инициализации. Например установка портов ввода-вывода в нужные направления и значения. Функция вызывается ПОСЛЕ загрузки кода программы в процессор. Затем определяется функция UserFinishCPUEnv, которая предназначена для совершения каких либо действий по окончании процесса программирования. Также возможно определение функции UserInitAfterReset, которая предназначена для каких либо инициализаций до загрузки кода в процессор, и до инициализации самого процессора (после его сброса JTAG-операцией CPU RESET), и UserInitAfterCPUInit – которая будет вызвана до загрузки кода, но после инициализации CPU. Две последние функции используются в случае сложных инициализаций, когда (например) для запуска того CPU, который будет программировать память, необходимо проинициализировать какие либо другие компоненты системы, например другой CPU. Такой пример можно найти в скрипте инициализации платы с процессором TMS320DM6446, где необходима определенная работа с ядром ARM для запуска ядра TMS320C64PLUS, которое уже занимается непосредственно программированием.

4.3. Скрипт-инициализатор процессора.

Скрипты этого назначения расположены в директории **<sauris_install_dir>/sauflash/init/cpu**. В скрипте-инициализаторе параметров процессора определяются параметры и функции, специфичные для процессора. Первый параметр, переменная cpu_type, строковой параметр, определяющий тип процессора. Каких либо ограничений на тип процессора нет, на сегодня задействованы TMS320VC33, TMS320C5500, TMS470R1B, TMS320C2400, TMS320C2800, TMS320C6410. Параметр в дальнейшем используется другими скриптами для определения семейства процессоров, с которым идет работа. Далее определяется список поддерживаемых микросхем памяти. На данный момент зарезервированы следующие значения: 0 – I2C EEPROM/I2C FRAM, 1 – parallel NOR Flash (Classic Flash), 2 – SPI EEPROM/SPI FLASH/SPI FRAM, 3 – Internal on-chip flash. Далее переменной FlashLoadSuppCode присваивается значение, определяющее путь к коду программы, обеспечивающий программирование памяти. Файл должен быть в формате COFF, производимом редактором связей (линкером) среды TI CCS или TI CC. Следующий параметр – TargetBigEndian. Особых пояснений не требует. 0 – Little endian, 1 – Big endian, 2 – при записи в память big endian, при чтении little (таким является например ARM7 в режиме BE, таковы особенности драйверов доступа к отладочным средствам ARM). Далее определяется название регистра счетчика команд. Переменная “pc_name” – как правило ей присваивается значение “PC”, так как в большинстве процессоров счетчик команд именуется именно так. Далее определяются базовые адреса контроллеров в адресном пространстве процессора. spiN_base, i2cN_base, emif_base, emifb_base, и т.п., а также переменная “word_width”, определяющая минимальный размер данного у процессора. Например для ARM или C64xx – это “0”, т.е. 8 бит, для C55xx или C28xx – “1”, т.е. 16 бит, а для VC33 – “2”, т.е. 32 бита. Для процессоров с внутренней памятью дополнительно определяются ее параметры – длины секторов, базовый адрес начала массива в адресном пространстве процессора, адрес контроллера флеш-памяти. Для некоторых процессоров (например 28xx) производятся вычисления тактовых частот и коэффициентов PLL из частоты кварца, определенной в инициализаторе платы. После чего, если требуется, определяется функция CPUSpecificInit, предназначенная для инициализации ядра процессора и его периферии для нормальной работы системы. Данная функция вызывается до загрузки кода в процессор. В ней производятся манипуляции с PLL и системой тактирования, разблокирование средств защиты кода, и т.п. Настройки.

4.4. Скрипт-инициализатор интерфейса.

Скрипты этого назначения расположены в директории **<sauris_install_dir>/sauflash/init/if**. В скрипте-инициализаторе параметров интерфейса определяются параметры, специфичные для интерфейса. Для большинства интерфейсов определяются переменные iftype (на данный момент: 0 – параллельная шина, напр. EMIF, 1 – I2C, 2 – MCBSP, 3 – SPI, 4 – условный интерфейс внутренней памяти (“internal”)), ctrlr_base, показывающий базовый адрес контроллера интерфейса в адресном пространстве процессора, и iface_init_script – имя скрипта, выполняемого для инициализации интерфейса и передачи его параметров программе в процессор. Эти скрипты расположены в поддиректории **supp** директории **if**. Для паралелльных интерфейсов определяются дополнительные параметры – xbus_width, ширина интерфейса целиком в битах; dbus_width, ширина той части шины, к которой подключена микросхема памяти (на широкую шину можно подключить несколько микросхем с более узкими шинами); flash_data_shift, показывающий, положение младшего бита шины микросхемы памяти на шине процессора; flash_base, показывающая начало блока памяти, представляемого этим интерфейсом в адресном пространстве процессора. Для «псевдо-интерфейса» к внутренней памяти указывается ширина слова этой памяти, переменная dbus_width.

4.5. Скрипт-инициализатор микросхемы памяти.

Скрипты этого назначения расположены в директории **<sauris_install_dir>/sauflash/init/ic**. В скрипте-инициализаторе параметров микросхемы памяти определяются параметры, специфичные для нее. Переменная flash_type определяет тип микросхемы (на данный момент 0 – I2C EEPROM/I2C FRAM, 1 – Parallel NOR Flash (Classic Flash), 2 – SPI EEPROM/SPI FLASH/SPI FRAM, 3 – Internal on-chip Flash). Далее определяется размер всей памяти в байтах, и тип и размер секторов (для стирания), при необходимости размер страниц (если микросхема программируется страницами). Для большинства микросхем памяти описание секторов производится следующим образом: Переменная uniform_sector определяет, все ли сектора одинакового размера, или нет. Далее, если uniform_sector==1, то в переменной sectorsize указывается размер сектора. Иначе этой же переменной присваивается список размеров секторов, в порядке от младшего адреса к старшему. Для внутренней памяти все эти параметры определяются тут же, но перерасчетом из параметров массива Flash, указанного в скрипте-инициализаторе процессора. Переменная “programming_by_byte” показывает, программируется данная микросхема постранично, или побайтово (пословно). И в конце определяется переменная “ ic_specific_setup”, показывающая путь к скрипту инициализации микросхемы и передачи ее параметров программе в процессор. Эти скрипты расположены в поддиректории **supp** директории **ic**. Так же могут определяться какие либо дополнительные параметры (например смещения для передачи команд для Parallel NOR Flash).

4.6. Исходные тексты программы для процессора, программирующего память, и функции, в них определенные.

hal.c

в данном файле определены следующие функции:

- uint16 GetByte (void* buf, uint16 ofs); - считывание 8-битного байта из буфера “buf” по смещению в 8-битных байтах ofs. Производит считывание байта независимо от endianess и минимального размера слова данных процессора.
- void PutByte (void* buf, uint16 ofs, uint16 byte); - запись 8-битного байта byte в буфера “buf” по смещению в 8-битных байтах ofs. Производит запись байта в буфер независимо от endianess и минимального размера слова данных процессора.
- uint32 GetWord(void* buf, uint32 ofs); Производит считывание слова, размер которого определен глобальной переменной “width”, начиная с любого смещения в 8-битных байтах ofs, независимо от endianess и минимального размера слова данных процессора.
- check_iface_params – проверяет корректность информации о параллельной шине.

data.c

в данном файле определены глобальные переменные, участвующие при обмене между PC и программой в процессоре.

classic.c

в этом файла определены функции для работы с классической параллельной NOR Flash памятью:

- static void Classic_Write(uint32 addr, uint32 word); Запись слова в микросхему памяти с учетом интерфейса, к которому она подключена, ширин шин, и остальных параметров подключения микросхемы. Addr – смещение в 8-битных байтах от начала микросхемы.
- uint32 Classic_Read(uint32 addr); Чтение слова из микросхемы памяти с учетом интерфейса, к которому она подключена, ширин шин, и остальных параметров подключения микросхемы. Addr – смещение в 8-битных байтах от начала микросхемы.
- static uint16 Classic_Wait(uint32 addr); Ожидание окончания процесса записи или стирания микросхемы.
- static void Classic_Reset(void); Сброс управляющего автомата микросхемы в исходное состояние.
- uint16 perform_classic_flash_program(); программирование блока данных из буфера “buffer” начиная со смещения (в 8-битных байтах относительно начала массива памяти ИМС) “address” и длиной (в 8-битных байтах) length с последующей верификацией. После успешного окончании программирования “address” должен быть увеличен на “length”.
- uint16 perform_classic_flash_erase(); Стирание сектора, в который попадает смещение “address” (в 8-битных байтах относительно начала массива памяти ИМС).

i2c-eeprom.c

в этом файла определены функции для работы с микросхемами с I2C интерфейсом:

- static uint16 send_i2c_byte (uint16 byte); отправка байта по интерфейсу I2C.
- static uint16 send_i2c_dblock(uint16 blksz, uint16 ofs, uint16 stop_flag); отправка блока данных по интерфейсу I2C.
- static uint16 recv_i2c_dblock(uint16 blksz, uint16 ofs); прием блока данных по интерфейсу I2C.
- static uint16 wait_for_i2c_EEPROM_ready(); Ожидание окончания процесса записи или стирания.
- uint16 perform_i2c_eeprom_program(); программирование блока данных из буфера “buffer” начиная со смещения (в 8-битных байтах относительно начала массива памяти ИМС) “address” и длиной (в 8-битных байтах) length с последующей верификацией. После успешного окончании программирования “address” должен быть увеличен на “length”.
- uint16 perform_i2c_eeprom_erase(); Стирание сектора (или страницы), в который попадает смещение “address” (в 8-битных байтах относительно начала массива памяти ИМС).

spi-eeprom.c

в этом файла определены функции для работы с микросхемами с SPI интерфейсом:

- static void SetCS(uint16 fActive); Установка сигнала CS в активное или неактивное состояние исходя из параметров порта этого сигнала, переданных из скриптов.
- static uint16 SPI_tx(uint16 data); Передача байта с последующим приемом. Обслуживаются как интерфейс SPI, так и MCBSP в режиме clock-stop.
- static void SendWREN(); Отправка команды WREN в EEPROM.
- static uint16 wait_for_SPI_EEPROM_ready(); Ожидание окончания процесса записи или стирания.
- static uint16 send_SPI_dblock(uint16 blksz, uint16 ofs, uint16 stop_flag); Запись блока данных из буфера “buffer” со смещения (в 8-битных байтах) “ofs” в микросхему памяти начиная со смещения от начала ее массива памяти (в 8-битных байтах) “address”.
- static uint16 recv_SPI_dblock(uint16 blksz, uint16 ofs); Считываение блока данных из микросхемы памяти со смещения от начала ее массива памяти (в 8-битных байтах) “address” в буфер “rcv_buf” со смещения (в 8-битных байтах) “ofs”
- uint16 perform_SPI_eeprom_program(); программирование блока данных из буфера “buffer” начиная со смещения (в 8-битных байтах относительно начала массива памяти ИМС) “address” и длиной (в 8-битных байтах) length с последующей верификацией. После успешного окончании программирования “address” должен быть увеличен на “length”.
- uint16 perform_SPI_eeprom_erase();Стирание сектора (или страницы), в который попадает смещение “address” (в 8-битных байтах относительно начала массива памяти ИМС).

int-flash???.c

в этих файлах определены функции для работы с внутренней памятью процессоров. Основные функции:

- uint16 perform_int_flash_program(); программирование блока данных из буфера “buffer” начиная со смещения (в 8-битных байтах относительно начала массива памяти) “address” и длиной (в 8-битных байтах) length с последующей верификацией. После успешного окончании программирования “address” должен быть увеличен на “length”.
- uint16 perform_int_flash_erase(); Стирание сектора (или страницы), в который попадает смещение “address” (в 8-битных байтах относительно начала массива памяти ИМС).

main.c

функция “main” программы. В ней производится инициализация переменных, которые будут переданы скрипту на PC (buffer_size, status), после чего начинается бесконечный цикл обработки команд, поступающих от PC. Первой операцией в цикле является команда программной точки останова, определяемая для каждого процессора по-своему. Для процессоров, у которых нет такой инструкции, должен быть определен глобальный символ “_BreakPoint”, по адресу которого скриптом будет установлена аппаратная точка останова. Далее, пока программа находится в состоянии останова, скрипт передает данные, команды и параметры, занося их в глобальные переменные программы. После чего продолжает исполнение программы. Программа, в зависимости от переданной команды и типа микросхемы памяти выполняет ту или иную функцию, после чего останавливается на той же точке останова. Скрипт считывает значение глобальной переменной “status”, определяя корректно ли была выполнена команда. И так повторяется до тех пор, пока скрипт не закончит свою работу.