Пакет расширения TCL XDS V1.0

Справочное руководство.

Table of Contents

1. Назначение.

Пакет XDS предоставляет доступ через язык скриптов TCL к отладочным и эмуляционным средствам процессоров фирмы Texas Instruments, подключенным к компьютеру при помощи JTAG-эмуляторов серии SAU510 фирмы SAURIS GmbH и к пограничному сканированию любых устройств, подключенных через данные эмуляторы.

2. Состав

Пакет XDS состоит из библиотеки **tclxds.dll**, исполняющей команды `xds` и `xdsutil`, и TCL-скрипта **xdshelp.tcl**, в котором описаны дополнительные функции, помогающие работать с командами `xds` и `xdsutil`. Пакет располагается в директории lib/tcl8.5/xds1.0 инсталляции TCL.

3. Подключение пакета. Завершение работы с пакетом.

Для использования команд `xds`, `xdsutil` и средств из **xdshelp.tcl** необходимо выполнить следующую команду:

```
package require xds
```

После завершения работы с пакетом необходимо выполнить следующую команду:

```
unload $TCLXDSLIBFULL
```

4. Переменные, определяемые после подключения пакета.

`TCLXDSLIB` – имя файла библиотеки расширения.
`TCLXDSLIBFULL` – полный путь к библиотеке расширения

5. *Команды пакета xds*

5.1. *Команды, предоставляемые библиотекой tclxds.dll*

5.1.1. *xds*

Команда предоставляет доступ к JTAG-эмулятору SAU510 и к устройствам, подключенным через него к компьютеру.

Формат команды:

```
xds <subcommand> ?parameters?
```

5.1.1.1 xds add

Формат команды:
`xds add device_name ?parent?`

Команда возвращает целое число, являющееся идентификатором устройства (ID) в дереве устройств. Данный идентификатор в дальнейшем будет использоваться во всех операциях с устройством.

Параметры:
`device_name` – имя устройства. Список поддерживаемых устройств и их параметров приведен далее, после описания команды **`xds param`**.
`parent` – идентификатор устройства-родителя. Если он не указан, то устройство добавляется в корень. Устройством-родителем могут быть исключительно устройства типа “JTAG-эмулятор” или “JTAG-маршрутизатор”. В корень могут быть добавлены исключительно устройства типа “JTAG-эмулятор”.

Описание команды:
Команда добавляет устройство в дерево устройств. Она предназначена для создания описания конфигурации отлаживаемого устройства. Устройство может быть подключено к компьютеру при помощи нескольких JTAG-эмуляторов, в JTAG-цепочке каждого из них может быть один или более компонентов, включая JTAG-маршрутизаторы, например ICEPICK-C, применяемый в процессорах OMAP и Davinci.

Пример использования:

```
package require xds
set emu0 [xds add SAU510]
set dev0 [xds add TMS320C2800 $emu0]
set dev1 [xds add GENERIC $emu0]
```

данный пример описывает два устройства, микроконтроллер семейства C28xx (для него доступна отладка, но не доступны JTAG SCAN операции), и второе устройство, не поддерживаемое пакетом как имеющее возможность отладки (для него доступны JTAG SCAN операции, но недоступна отладка), подключенные к компьютеру через эмулятор серий SAU510. Для этого в корень добавлен эмулятор, а TMS320C2800 и GENERIC добавлены с указанием в качестве родителя ID-а эмулятора.

5.1.1.2 xds param

Формат команды:
`xds param ID param_name param_value`

Команда не возвращает значений.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**
`param_name` – имя параметра. Список поддерживаемых устройств и их параметров приведен далее. Формат имени параметра всегда «section_name.param_name», где section_name и param_name это слова, не содержащие знака «.»
`param_value` – значение параметра.

Описание команды:
Команда устанавливает устройству с идентификатором **`ID`** параметр с именем **`param_name`** и значением **`param_value`**. Она предназначена для конфигурирования устройств.

Пример использования:

```
package require xds
#
set emu0 [xds add SAU510]
set dev0 [xds add TMS320C2800 $emu0]
set dev1 [xds add GENERIC $emu0]
#
xds param $emu0 sepk.pod_tckdiv 10
xds param $dev1 generic.irbits 8
xds param $dev1 generic.drbits 1
```

Данный скрипт устанавливает устройству с идентификатором из переменной emu0 параметр “sepk.pod_tckdiv” со значением 10, устройству с идентификатором из переменной dev1 параметры “irbits” со значением 8 и “drbits” со значением 1.

Список поддерживаемых устройств и их параметров:

SAU510, SAU510ISO– Эмуляторы SAU510-USB, SAU510-USB Plus (**SAU510**) или SAU510-USB Iso Plus (**SAU510ISO**)
Списки параметров для соответствующих эмуляторов можно посмотреть в скриптах sau510.tcl и sau510iso.tcl, находящихся в директории sauflash/init/emu в инсталляционном директории, указанном при инсталляции. Описание значений этих параметров полностью соответствует таковому для конфигурирования эмуляторов через “additional board configuration file” при конфигурировании среды TI Code Composer Studio.

GENERIC – любое устройство в JTAG цепочке.
Устройства типа GENERIC могут иметь два параметра: generic.irbits и generic.drbits, соответственно характеризующих длину регистра IR в битах, и длину регистра DR,

когда в IR загружена инструкция BYPASS (обычно длина DR 1 бит). Данная информация может быть получена из BSDL-модели микросхемы. Параметр irbits по умолчанию равен 2, параметр drbits опциональный, по умолчанию равен 1.

ICEPICK-A, ICEPICK-B, ICEPICK-C — JTAG-маршрутизаторы. Все они обладают одним обязательно указываемым параметром «router.subpaths N», показывающий количество задействованных выходов маршрутизатора.

PORT — порт маршрутизатора. Имеет параметры «router.address N», «router.custom no/yes» и «router.default no/yes». Параметр address обязателен, и показывает номер порта у маршрутизатора, параметры custom и default для маршрутизаторов ICEPICK должны быть указаны и установлены в «no»

ARM7 – Микропроцессор ARM7. Параметров нет.
ARM9 – Микропроцессор ARM9. Параметров нет.
TMS320C2400 – Микроконтроллер серии C24xx. Параметров нет.
TMS320C2800 – Микроконтроллер серии C28xx. Параметров нет.
TMS320C5500 – Микропроцессор серии C55xx. Параметров нет.
TMS320VC33 – Микропроцессор TMS320VC33. Параметров нет.
TMS320C6410 – Микропроцессор серии TMS320C64xx с ядром ревизии 1.1. Таковыми являются большинство процессоров 64xx, которые не на ядре C64+. Например TMS320C6416 или DSP-ядро в TMS320DM642. Параметров нет.
TMS320C64PLUS – Микропроцессор серии TMS320C64+, включая C674x
TMS320C6200 – Микропроцессор серии C620x. Параметров нет.
TMS320C6210 – Микропроцессор серии C621x. Параметров нет.
TMS320C6220 – Микропроцессор серии C622x. Параметров нет.
TMS320C6700 – Микропроцессор серии C670x. Параметров нет.
TMS320C6710 – Микропроцессор серии C671x. Параметров нет.
TMS320C6720 – Микропроцессор серии C672x. Параметров нет.

Примечание к **`xds add`** и **`xds param`**.

В корне дерева могут находится только эмуляторы. Т.е. Устройства типа «SAU510». Все остальные устройства должны добавляться с указанием родителя. Родителем для устройства, представляющего собой целевое отлаживаемое устройство (например микропроцессор), либо маршрутизатора, может быть только либо эмулятор, либо порт маршрутизатора. Родителем для порта маршрутизатора может быть только маршрутизатор.

Пример описания JTAG-цепочки системы на кристалле OMAP-L137:

JTAG-цепочка SoC ***OMAP-L137*** представляет собой следующее дерево:

```
SAU510 Emulator
└── ICEPICK-C router  (subpaths=3)
    ├── Port #2   (address=19, custom=no, default=no)
    │   └── GENERIC (irbits=4)  (ETB-11, непосредственно не поддержан)
    ├── Port #1   (address=18, custom=no, default=no)
    │   └── ARM9
    └── Port #0   (address=17, custom=no, default=no)
        └── TMS320C64PLUS
```

TCL-скрипт, описывающий эту цепочку, подключенную к эмулятору SAU510ISO, выглядит следующим образом:

```
package require xds

set EMU [xds add SAU510ISO]

# TCK generator parameters
xds param $EMU sepk.pod_tckdiv 10
xds param $EMU sepk.pod_tckexp 0
xds param $EMU sepk.pod_tckpredivena NO
xds param $EMU sepk.pod_no_tckr NO
xds param $EMU sepk.pod_tck_bpas_dly 0
xds param $EMU sepk.pod_aclk_enable YES
# Terminators
xds param $EMU sepk.pod_tckload YES
# TMS/TDO timing
xds param $EMU sepk.pod_tdoontckfall NO
xds param $EMU sepk.pod_tms_ofs 0
xds param $EMU sepk.pod_tdo_ofs 0
# Link delay
xds param $EMU uscif.linkdly 3
# router's (icepick, DAP) parameters
xds param $EMU router.adaptive_tclk NO
xds param $EMU router.skip_polling YES
# other, required, don't change
xds param $EMU sepk.pod_emuversion 2
xds param $EMU sepk.pod_slowclk NO
xds param $EMU sepk.pod_aw_en YES
xds param $EMU sepk.pod_scanloop NO
xds param $EMU uscif.clkmode normal

set router [xds add ICEPICK-C $EMU]
xds param $router router.subpaths 3

set etb_port [xds add PORT $router]
xds param $etb_port router.address 19
xds param $etb_port router.default no
xds param $etb_port router.custom no

set arm_port [xds add PORT $router]
xds param $arm_port router.address 18
xds param $arm_port router.default no
xds param $arm_port router.custom no

set dsp_port [xds add PORT $router]
xds param $dsp_port router.address 17
xds param $dsp_port router.default no
xds param $dsp_port router.custom no

set DSP   [xds add TMS320C64PLUS $dsp_port]
set ARM   [xds add ARM9 $arm_port]
set ETB11 [xds add GENERIC $etb_port]
xds param $ETB11 generic.irbits 4
```

после выполнения указанных в примере команд идентификаторы устройств из переменных $DSP, $ARM и $ETB11 можно использовать для дальнейшей работы с этими частями SoC.

Список поддерживаемых устройств постоянно расширяется, также он может быть расширен и по просьбе пользователя. Запросы на расширение следует направлять на e-mail sauris_support@scanti.ru

5.1.1.3 xds connect

Формат команды:
`xds connect ID`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**. При этом устройство, к которому производится подключение не может быть устройством типа «JTAG эмулятор», так как эмулятор лишь обеспечивает подключение к устройству, но сам не может быть подключен.

Описание команды:
Команда устанавливает подключение к устройству с идентификатором **`ID`**. Подключение к устройству аналогично действию “Connect” в среде Code Composer Studio. После подключения к устройству становятся доступны все остальные функции, обеспечивающие доступ к данному устройству. Если устройство является микропроцессором, то в результате подключения он переходит в отладочный (эмуляционный) режим и переводится в состояние останова (HALT). Для остальных типов устройств изменения их состояния не происходит.

Пример использования:

```
package require xds
#
source “init\emu\sau510iso.tcl”
source “init\emu\my5501brd.tcl”
#
set result [xds connect $DEV]
if { $result != 0 } {
  puts “Error connecting to target”
  puts [xds fetch_errors $DEV]
}
```

Данный скрипт устанавливает соединение с устройством с идентификатором в переменной $DEV, которая определяется в скриптах, вызванных командами **`source`**. В этих же скриптах конфигурируется устройство.

5.1.1.4 xds reset

Формат команды:
`xds reset ID`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда производит сброс устройства с идентификатором **`ID`**. Действие команды аналогично действию “Reset CPU” в среде Code Composer Studio. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
package require xds
#
source “init\emu\sau510iso.tcl”
source “init\emu\my5501brd.tcl”
#
set result [xds connect $DEV]
if { $result != 0 } {
  puts “Error connecting to target”
  puts [xds fetch_errors $DEV]
}
set result [xds reset $DEV]
if { $result != 0 } {
  puts "Error resetting target CPU"
}
```

Данный скрипт устанавливает соединение с устройством с идентификатором в переменной $DEV, которая определяется в скриптах, вызванных командами **`source`**. После чего производит сброс устройства $DEV.

5.1.1.5 xds run

Формат команды:
`xds run ID`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда производит запуск устройства с идентификатором **`ID`** на исполнение инструкций. Действие команды аналогично действию “Run” в среде Code Composer Studio. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
set result [xds run $DEV]
if { $result != 0 } {
  puts "Error running DSP code”
}
```

Данный скрипт запускает устройство с идентификатором $DEV на выполнение кода.

5.1.1.6 xds halt

Формат команды:
`xds halt ID`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда производит останов устройства с идентификатором **`ID`**. Действие команды аналогично действию “Halt” в среде Code Composer Studio. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
set result [xds halt $DEV]
if { $result != 0 } {
  puts "Error stopping DSP”
}
```

Данный скрипт переводит устройство с идентификатором $DEV в состояние останова.

5.1.1.7 xds step

Формат команды:
`xds step ID ?count?`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`?count?` **–** Необязательный параметр, показывающий число шагов.

Описание команды:
Команда производит пошаговый проход программы в устройстве с идентификатором **`ID`**. После исполнения команды процессор исполняет одну инструкцию (или **`?count?`** инструкций). Действие команды аналогично действию “Step” в среде Code Composer Studio при отладке в режиме дизассемблера. Команда не может быть применена к устройствам типа GENERIC. Команда должна применяться к устройствам, находящимся в состоянии останова.

Пример использования:

```
#...
set result [xds step $DEV]
if { $result != 0 } {
  puts "Error performing single step on DSP”
}
```

Данный скрипт выполняет одну инструкцию в устройстве с идентификатором $DEV.

5.1.1.8 xds setbp

Формат команды:
`xds setbp ID address mempage`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`address` **–** Адрес в адресном пространстве процессора.
`mempage` **–** Тип адресного пространства (значения соответствуют используемым в PAGE в .cmd файле линкера для данного процессора). Например для TMS320VC55xx mempage=0 соответствует code space и байтовым адресам, а mempage=2 – IO Space и словесным адресам.

Описание команды:
Команда производит установку точки останова выполнения инструкций. mempage должна соответствовать пространству кода процессора, как правило это число 0. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
set result [xds setbp $DEV 0x1234 0]
if { $result != 0 } {
  puts "Error setting breakpoint”
}
```

Данный скрипт устанавливает точку останова в устройстве с идентификатором $DEV по адресу 0x1234.

5.1.1.9 xds clearbp

Формат команды:
xds clearbp ID address mempage

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
ID – Идентификатор устройства, полученный от команды **xds add**, к которому было произведено подключение по команде xds connect.
address – Адрес в адресном пространстве процессора.
mempage – Тип адресного пространства (значения соответствуют используемым в PAGE в .cmd файле линкера для данного процессора). Например для TMS320VC55xx mempage=0 соответствует code space и байтовым адресам, а mempage=2 – IO Space и словесным адресам.

Описание команды:
Команда производит снятие точки останова выполнения инструкций, установленной перед этим командой **xds setbp**. **address** и mempage должны точно соответствовать указанным в **xds setbp**. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
set result [xds clearbp $DEV 0x1234 0]
if { $result != 0 } {
  puts "Error clearing breakpoint”
}
```

Данный скрипт снимает точку останова в устройстве с идентификатором $DEV по адресу 0x1234.

5.1.1.10 xds resetbp

Формат команды:
`xds resetbp ID`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда производит снятие всех точек останова, установленной перед этим командами **`xds setbp`**. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
set result [xds resetbp $DEV]
if { $result != 0 } {
  puts "Error resetting all breakpoints”
}
```

Данный скрипт снимает все точки останова в устройстве с идентификатором $DEV.

5.1.1.11 xds read_reg

Формат команды:
`xds read_reg ID reg_name`

Команда возвращает строку “0 <hexdata> <length>” при успешном завершении, и число, отличное от 0 при ошибке. <hexdata> - значение, считанное из регистра в шестнадцатеричном виде без каких либо префиксов. <length> - длина значения в битах.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`reg_name` **–** Наименование регистра устройства.

Описание команды:
Команда производит чтение значения из регистра устройства. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
puts [xds read_reg $DEV PC]
```

Данный скрипт выводит на экран результат считывания содержимого регистра “PC” (счетчика команд).

5.1.1.12 xds write_reg

Формат команды:
`xds write_reg ID hexdata reg_name`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`hexdata` **–** Значение для записи в регистр. Шестнадцатеричное число без каких либо префиксов (формат <hexdata>).
`reg_name` **–** Наименование регистра устройства.

Описание команды:
Команда производит запись значения в регистра устройства. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
set result [xds write_reg $DEV 123ABC PC]
if { $result != 0 } {
  puts "Error setting PC register to 0x123ABC”
}
```

Данный скрипт заносит значение 0x123ABC в регистр PC (счетчик команд).

5.1.1.13 xds read_mem

Формат команды:
`xds read_mem ID address length mempage`

Команда возвращает строку “0 <hexdata> <length>” при успешном завершении, и число, отличное от 0 при ошибке. <hexdata> - значение, считанное из памяти в шестнадцатеричном виде без каких либо префиксов, представляющее собой единое <length>-битное число в виде MSB…LSB, где LSB – первый бит с начала считываемого блока памяти. <length> - длина значения в битах.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`address` **–** Адрес начала блока памяти
`length` **–** Длина блока памяти в 8-битных байтах
`mempage` **–** Тип адресного пространства (значения соответствуют используемым в PAGE в .cmd файле линкера для данного процессора). Например для TMS320VC55xx mempage=0 соответствует code space и байтовым адресам, а mempage=2 – IO Space и словесным адресам.

Описание команды:
Команда производит чтение блока памяти из адресного пространства устройства. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
puts [xds read_mem $DEV 0xFFFF80 8 0]
```

Данный скрипт выводит на экран результат считывания содержимого блока памяти длиной 8 8-битных байт, расположенного с адреса 0xFFFF80 в странице (типе адресного пространства) 0.

5.1.1.14 xds write_mem

Формат команды:
`xds write_mem ID address length mempage hexdata`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`address` **–** Адрес начала блока памяти
`length` **–** Длина блока памяти в 8-битных байтах
`mempage` **–** Тип адресного пространства (значения соответствуют используемым в PAGE в .cmd файле линкера для данного процессора). Например для TMS320VC55xx mempage=0 соответствует code space и байтовым адресам, а mempage=2 – IO Space и словесным адресам.
`hexdata` **–** значение для записи в память в шестнадцатеричном виде без каких либо префиксов, представляющее собой единое <length>-битное число в виде MSB…LSB, где LSB – первый бит с начала считываемого блока памяти.

Описание команды:
Команда производит запись значения hexdata в блок памяти в адресном пространстве устройства. Команда не может быть применена к устройствам типа GENERIC.

Пример использования:

```
#...
set result [xds write_mem $DEV 0x1000 6 1 123456789ABC]
if { $result != 0 } {
  puts "Error writing 6-byte memory block”
}
```

Данный скрипт заносит в память начиная с адреса 0x1000 в адресном пространстве PAGE=1 блок из 6 8-битных байт (48-бит), представленных шестнадцатеричным числом **`123456789ABC`**

5.1.1.15 Команда xds status

Формат команды:
`xds status ID`

Команда возвращает строку “0 <hexdata> <length>” при успешном завершении, и число, отличное от 0 при ошибке. <hexdata> - значение, считанное из регистра текущего состояния эмуляционной системы процессора в шестнадцатеричном виде без каких либо префиксов. <length> - длина значения в битах.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда производит чтение регистра статуса эмуляционной системы процессора.
Команда не может быть применена к устройствам типа GENERIC.
Основные биты данного регистра:
0x01 – Процессор запущен под контролем эмулятора
0x02 – сработала точка останова
0x04 – выполнен некорректный код инструкции
0x20 – пользовательский останов
0x40 – процессор в состоянии останова.
0x80 – Процессор запущен без контроля эмулятором
0x100 – Выполнение инструкции не завершено
0x200 – Некорректное состояние процессора

Пример использования:

```
#...
puts [xds status $DEV]
```

Данный скрипт выводит на экран результат считывания содержимого регистра статуса эмуляционной системы процессора.

5.1.1.16 xds disconnect

Формат команды:
`xds disconnect ID`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда разрывает подключение к устройству с идентификатором **`ID`**. Подключение к устройству аналогично действию “Disconnect” в среде Code Composer Studio.

Пример использования:

```
package require xds
#
source “init\emu\sau510iso.tcl”
source “init\emu\my5501brd.tcl”
#
set result [xds connect $DEV]
if { $result != 0 } {
  puts “Error connecting to target”
  puts [xds fetch_errors $DEV]
}
set result [xds disconnect $DEV]
if { $result != 0 } {
  puts “Error disconnecting from target”
  puts [xds fetch_errors $DEV]
}
```

Данный скрипт устанавливает соединение с устройством с идентификатором в переменной $DEV, которая определяется в скриптах, вызванных командами **`source`**, а затем его разрывает.

5.1.1.17 xds quit

Формат команды:
xds quit ID

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
ID – Идентификатор устройства, полученный от команды **xds add**, к которому хоть раз было произведено подключение по команде xds connect и последующее отключение по команде xds disconnect

Описание команды:
Команда переводит внутренние структуры в начальное состояние, которое было перед самым первым подключением к устройству, освобождая занятую память, и т.п..

Пример использования:

```
package require xds
#
source “init\emu\sau510iso.tcl”
source “init\emu\my5501brd.tcl”
#
set result [xds connect $DEV]
if { $result != 0 } {
  puts “Error connecting to target”
  puts [xds fetch_errors $DEV]
}
set result [xds disconnect $DEV]
if { $result != 0 } {
  puts “Error disconnecting from target”
  puts [xds fetch_errors $DEV]
}
xds quit $DEV
unload $TCLXDSLIBFULL
```

Данный скрипт устанавливает соединение с устройством с идентификатором в переменной $DEV, которая определяется в скриптах, вызванных командами **source**, а затем его разрывает и корректно завершает работу.

5.1.1.18 xds fetch_errors

Формат команды:
xds fetch_errors ID

Команда возвращает список внутренних кодов ошибок, накопленных в драйвере устройства.

Параметры:
ID – Идентификатор устройства, полученный от команды **xds add**, к которому хоть раз было произведено подключение по команде xds connect

Описание команды:
Команда возвращает список ошибок, накопленных в драйвере устройства и сбрасывает признак ошибки

Пример использования:

```
package require xds
#
source “init\emu\sau510iso.tcl”
source “init\emu\my5501brd.tcl”
#
set result [xds connect $DEV]
if { $result != 0 } {
  puts “Error connecting to target”
  puts [xds fetch_errors $DEV]
}
```

В случае ошибки подключения будет выведен на экран список кодов внутренних ошибок драйвера.

5.1.1.19 xds acquire

Формат команды:
`xds acquire ID`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда захватывает управление устройством типа GENERIC. При этом настраивается JTAG-цепочка и все маршрутизаторы, находящиеся в ней на доступ к данному устройству, а остальные устройства переводятся в BYPASS. После выполнения этой команды **возможна работа только с устройством**, чей ID был указан в команде. Без выполнения этой операции команды сканирования (state и scan) будут захватывать и освобождать устройство автоматически до и после операции соответственно. Команда предназначена для захвата управления устройством для выполнения на нем нескольких команд сканирования подряд с гарантией непрерывности. Данная команда может быть выполнена только для устройств типа GENERIC (для отлаживаемых устройств операции Acquire и Release делаются всегда автоматически в процессе выполнения более высокоуровневых команд).

Пример использования:

```
#...
set result [xds acquire $DEV]
if { $result != 0 } {
  puts “Error acquiring device”
}
```

Данный скрипт захватывает управление устройством типа GENERIC с идентификатором $DEV.

5.1.1.20 xds release

Формат команды:
`xds release ID`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда освобождает устройство типа GENERIC, управление которым было захвачено по **`xds acquire`**. Данная команда может быть выполнена только для устройств типа GENERIC.

Пример использования:

```
#...
set result [xds release $DEV]
if { $result != 0 } {
  puts “Error releasing device”
}
```

Данный скрипт освобождает устройство типа GENERIC с идентификатором $DEV.

5.1.1.21 xds state

Формат команды:
`xds state ID jtag_state`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`jtag_state` – Одно из состояний автомата JTAG, в которое он будет переведен после выполнения команды. Возможные варианты {tlr runtest dpause ipause} что соответствует TEST-LOGIC-RESET, RUNTEST/IDLE, DPAUSE и IPAUSE.

Описание команды:
Команда переводит автомат JTAG устройства типа GENERIC в состояние jtag_state. Перевод в промежуточные состояния невозможен, так как у эмуляторов типа XDS510 сигнал TCK не останавливается. Данная команда может быть выполнена только для устройств типа GENERIC.

Пример использования:

```
#...
set result [xds state $DEV runtest]
if { $result != 0 } {
  puts “Error setting device JTAG state”
}
```

Данный скрипт переводит автомат JTAG устройства типа GENERIC с идентификатором $DEV в состояние RUNTEST/IDLE.

5.1.1.22 xds scan

Формат команды:
`xds scan ID reg length data jtag_state ?option? ?option?`

Команда возвращает 0 при успешном завершении, и не 0 при ошибке.

Параметры:
`ID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`reg` – регистр для операции сканирования. “ir” или “dr”.
`length` – длина операции сканирования в битах.
`data` – данные для сканирования (подачи на TDI) в формате <hexdata> (см. описание read_mem/write_mem; аналогично представлению данных в формате SVF). Вместо данных возможно указание слов “ones”, “zeroes” или “circulate”, что означает сканирование одних лог. единиц, нулей или замыкание выхода TDO на вход TDI.
`jtag_state` – Одно из состояний автомата JTAG, в которое он будет переведен после выполнения команды. Возможные варианты {tlr runtest dpause ipause} что соответствует TEST-LOGIC-RESET, RUNTEST/IDLE, DPAUSE и IPAUSE.
`?option?` – необязательные параметры - опции. Если указать опцию “sendrecv”, то в процессе сканирования будут приняты данные с TDO и возвращены командой в виде “hexdata”, без этой опции данные с TDO будут проигнорированы. Если указать опцию “immediate”, то команда (если не было опции “sendrecv”) выполнится немедленно. Если не указать – то выполнение может быть отложено до следующей команды, которая требует приема данных (в целях оптимизации скорости).

Описание команды:
Команда производит операцию JTAG сканирования. Данная команда может быть выполнена только для устройств типа GENERIC.

Пример использования:

```
#...
puts [xds scan $DEV ir 8 5A dpause immediate]
```

Данный скрипт производит операцию сканирования 8 бит в регистр IR устройства, значения бит определяются числом 0x5A, после операции автомат JTAG переводится в состояние DPAUSE, и операция выполняется немедленно.

ВНИМАНИЕ! Операция сканирования производится не на всей JTAG-цепочке в целом, а на отдельно взятом выбранном устройстве. Преамбулы и постамбулы рассчитываются и отправляются автоматически, управление маршрутизаторами производится автоматически, все остальные устройства переводятся в BYPASS автоматически. Таким образом указанные длины регистров должны ТОЧНО СООТВЕТСТВОВАТЬ реальным значениям для устройства в текущем режиме, иначе последствия не предсказуемы. Также как и

должны быть корректно заданы параметры irbits и drbits для всех GENERIC-устройств в конфигурации.

5.1.2. ***xdsutil***

Команда предоставляет дополнительные функции, связанные с работой с командой xds. В том числе разбор структур COFF-файлов (исполняемых модулей для микропроцессоров TI), преобразование бинарных данных в формат hexdata и обратно.

Формат команды:

```
xdsutil <subcommand> ?parameters?
```

5.1.2.1 xdsutil bintohexdata

Формат команды:
`xdsutil bintohexdata binary_data`

Команда возвращает строку со значением вида <hexdata>, образованную из объекта с двоичными данными.

Параметры:
`Binary_data` – входные двоичные данные.

Описание команды:
Команда преобразует массив байт (объект двоичных данных binary_data) в формат hexdata.

Пример использования:

```
#...
fconfigure $fp -translation binary
set bindata [read $fp 1024]
set data [xdsutil bintohexdata $bindata]
```

Данный скрипт считывает из файла 1024 байт данных и преобразует их в формат hexdata для операций сканирования или записи в память.

5.1.2.2 xdsutil bintohexdata

Формат команды:
`xdsutil hexdatatobin hexdata ?length?`

Команда возвращает двоичный объект с данными, полученными преобразованием строки со значением вида <hexdata> в двоичный вид.

Параметры:
`hexdata` – данные в формате hexdata.
`length` – Необязательный параметр. Длина требуемого объекта двоичных данных (не менее, чем требуется байт для размещения hexdata целиком).

Описание команды:
Команда преобразует данные в формате hexdata в массив байт (объект двоичных данных). В случае указания length размер выходного объекта дополняется нулевыми данными до длины length.

Пример использования:

```
#...
set bindata [xdsutil bintohexdata 1234CDEF87]
```

Данный скрипт преобразует строку “hexdata” 1234CDEF в набор байт со значениями 87 EF CD 34 12.

5.1.2.3 xdsutil coff

Команда предоставляет функции по разбору структур COFF-файлов (исполняемых модулей для микропроцессоров TI).

Формат команды:

```
xdsutil coff <subcommand> ?parameters?
```

5.1.2.3.1 *xds coff load*

Формат команды:
`xdsutil coff load path`

Команда возвращает ID считанного COFF-файла.

Параметры:
`path` – путь к COFF-файлу.

Описание команды:
Команда считывает COFF-файл и подготавливает его для работы остальных функций, возвращая его ID. Команды выгрузки не существует, память освобождается при операции unload пакету.

Пример использования:

```
#...
set coff [xdsutil coff load “./myproject/release/test.out”]
```

Данный скрипт считывает файл “./myproject/release/test.out” и возвращает его ID, присваивая его переменной “coff” .

5.1.2.3.2 *xds coff flags*

Формат команды:
`xdsutil coff flags coffID`

Команда возвращает список флагов считанного COFF-файла.

Параметры:
`coffID` – ID COFF-файла, полученный от команды **`xdsutil coff load`**

Описание команды:
Команда возвращает список флагов из заголовка COFF-файла. В том числе семейство процессоров, под которое сделан этот COFF (в виде FAMILY=<family>), точку входа (ENTRYPOINT=xxxxx), и опции, например EXEC или RELFLG

Пример использования:

```
#...
puts [xdsutil coff flags $coff]
```

Данный скрипт выводит в консоль список флагов COFF файла с ID из переменной coff.

5.1.2.3.3 *xds coff sections*

Формат команды:
`xdsutil coff sections coffID`

Команда возвращает список секций считанного COFF-файла.

Параметры:
`coffID` – ID COFF-файла, полученный от команды **`xdsutil coff load`**

Описание команды:
Команда возвращает список секций COFF-файла.

Пример использования:

```
#...
puts [xdsutil coff sections $coff]
```

Данный скрипт выводит в консоль список секций COFF файла с ID из переменной coff.

5.1.2.3.4 *xds coff symbols*

Формат команды:
`xdsutil coff symbols coffID`

Команда возвращает список символов считанного COFF-файла.

Параметры:
`coffID` – ID COFF-файла, полученный от команды **`xdsutil coff load`**

Описание команды:
Команда возвращает список символов COFF-файла.

Пример использования:

```
#...
puts [xdsutil coff symbols $coff]
```

Данный скрипт выводит в консоль список символов COFF файла с ID из переменной coff.

5.1.2.3.5 *xds coff sectionflags*

Формат команды:
`xdsutil coff sectionflags coffID section`

Команда возвращает список флагов указанной секци считанного COFF-файла.

Параметры:
`coffID` – ID COFF-файла, полученный от команды **`xdsutil coff load`**
`section` – имя секции COFF файла.

Описание команды:
Команда возвращает список флагов секции COFF-файла. Часть информации возвращается в виде “NAME=VALUE”, например адреса расположения и длина, а часть в виде “NAME”, например DATA, DSECT или NOLOAD

Пример использования:

```
#...
set sections [xdsutil coff sections $coff]
foreach section $sections {
  puts –nonewline “$section :”
  puts [xdsutil coff sectionflags $coff $section]
}
```

Данный скрипт выводит в консоль список секций COFF файла с указанием флагов для каждой секции. Т.е. отчет вроде карты загрузки файла.

5.1.2.3.6 *xds coff symbolflags*

Формат команды:
`xdsutil coff symbolflags coffID symbol`

Команда возвращает список флагов указанного символа считанного COFF-файла.

Параметры:
`coffID` – ID COFF-файла, полученный от команды **`xdsutil coff load`**
`symbol` – имя символа COFF файла.

Описание команды:
Команда возвращает список флагов символа COFF-файла. Часть информации возвращается в виде “NAME=VALUE”, например адрес, а часть в виде “NAME”, например C_FILE или ABSOLUTE.

Пример использования:

```
#...
set syms [xdsutil coff symbols $coff]
foreach sym $syms {
  puts –nonewline “$sym :”
  puts [xdsutil coff setionflags $coff $sym]
}
```

Данный скрипт выводит в консоль список всех символов COFF файла с указанием их параметров.

5.1.2.3.7 *xds coff sectiondata*

Формат команды:
`xdsutil coff sectiondata coffID section`

Команда возвращает двоичный объект данных с данными, предназначенными для загрузки этой секции в процессор.

Параметры:
`coffID` – ID COFF-файла, полученный от команды **`xdsutil coff load`**
`section` – имя секции COFF файла.

Описание команды:
Команда возвращает двоичный объект данных с данными, предназначенными для загрузки этой секции в процессор.

Пример использования:

```
#...
set sections [xdsutil coff sections $coff]
foreach section $sections {
  set bindata [xdsutil coff sectiondata $coff $section]
#...
}
```

Данный скрипт присваивает переменной bindata по очереди объекты двоичных данных, соответствующий данным из всех секций COFF файла.

5.2. TCL-процедуры, определенные в xdshelp.tcl

5.2.1. ldpgm

Формат команды:
ldpgm coffID devID

Команда не возвращает данных.

Параметры:
coffID – ID COFF-файла, полученный от команды **xdsutil coff load**
devID – Идентификатор устройства, полученный от команды **xds add**, к которому было произведено подключение по команде xds connect.

Описание команды:
Команда загружает COFF файл с заданным ID в устройство с заданным ID через JTAG-интерфейс.

Пример использования:

```
#...
set coff [xdsutil coff load “./myproject/release/test.out”]
ldpgm $coff $DEV
```

Данный скрипт загружает в устройство $DEV COFF-файл ./myproject/release/test.out

5.2.2. ***xerror***

Формат команды:
`xerror devID errmsg exit_code`

Команда не возвращает данных.

Параметры:
`devID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`errmsg` – сообщение об ошибке
`exit_code` – код завершения.

Описание команды:
Команда выводит сообщение о ошибке, затем вызывает **`xds fetch_errors`**, выводя результат в консоль, и корректно завершает работу скрипта (при условии, что подсоединено было всего одно устроство devID) с выходом в ОС. Применяется для сообщении о фатальной ошибке в результате обращения к устройству.

Пример использования:

```
#...
xerror $DEV “Error setting register value” 1
```

Данный скрипт выводит сообщение об ошибке, затем коды внутренних ошибок драйвера, и затем завершает работу скрипта с выходом в ОС с exit code = 1

5.2.3. *ferror*

Аналогична xerror, за исключением отсутствия вызова `xds fetch_errors` и вывода результата в консоль.

5.2.4. ***parmvalue***

Формат команды:
`parmvalue flags name`

Команда возвращает значение параметра с именем name из списка флагов секции, символа или COFF-файла и используется для флагов в формате NAME=VALUE.

Параметры:
`flags` – список флагов, например результат команды `xdsutil coff sectionflags`.
`name` – имя параметра.

Описание команды:
Команда возвращает значение параметра с именем name из списка флагов секции, символа или COFF-файла и используется для флагов в формате NAME=VALUE.

Пример использования:

```
#...
puts [parmvalue [xdsutil coff flags $coff] ENTRYPOINT]
```

Данный скрипт выводит значение точки входа, полученное из COFF файла с ID из переменной coff

5.2.5. ***getsym***

Формат команды:
`getsym coffID name`

Команда создает переменные вида “name_addr” и “name_pg”, в которые заносит значения адреса символа из COFF-файла и его страницы (PAGE).

Параметры:
`coffID` – ID COFF-файла, полученный от команды **`xdsutil coff load`**
`name` – имя символа.

Описание команды:
Команда создает переменные с именами “name_addr” и “name_pg”, в которые заносит значения адреса символа из COFF-файла и его страницы (PAGE).

Пример использования:

```
#...
getsym $coff _buffer
```

Данный скрипт создает две переменные “_buffer_addr” и “_buffer_pg”, в которые заносит значения адреса и страницы символа “_buffer” (он соответствует переменной “buffer” в С-коде, из которого был получен COFF-файл.

5.2.6. *PutShort u PutLong*

Формат команды:
`Put<Long/Short> devID addr pg data`

Команда не возвращает данных.

Параметры:
`devID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`addr` – адрес в адресном пространстве устройства devID
`pg` – страница адресного пространства устройства devID (аналогично PAGE .cmd файла линкера)
`data` – данные для занесения по указанному адресу.

Описание команды:
Команда заносит в память процессора 32- или 16-битное число data (обычный числовой формат, не hexdata) с учетом Target Endianess, которая анализируется на основе переменной нулевого уровня вложенности процедур “ TargetBigEndian”

Пример использования:

```
#...
PutShort $DEV 0x1000 0 0x1234
```

Данный скрипт заносит в память устройства $DEV по адресу 0x1000 PAGE 0 число 0x1234 длиной 16 бит.

5.2.7. *ReadShort u ReadLong*

Формат команды:
Read<Long/Short> devID addr pg

Команда возвращает считанное из памяти устройства данное в обычном числовом формате (Int, не hexdata).

Параметры:
devID – Идентификатор устройства, полученный от команды **xds add**, к которому было произведено подключение по команде xds connect.
addr – адрес в адресном пространстве устройства devID
pg – страница адресного пространства устройства devID (аналогично PAGE .cmd файла линкера)

Описание команды:
Команда возвращает считанное из памяти устройства данное в обычном числовом формате (Int, не hexdata) с учетом Target Endianess, которая анализируется на основе переменной нулевого уровня вложенности процедур “ TargetBigEndian”

Пример использования:

```
#...
puts [ReadShort $DEV 0x1000 0]
```

Данный скрипт выводит в консоль значение из памяти устройства $DEV по адресу 0x1000 PAGE 0 длиной 16 бит.

5.2.8. ***WriteReg***

Формат команды:
`WriteReg devID name data`

Команда не возвращает данных.

Параметры:
`devID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`name` – имя регистра устройства
`data` – данные для занесения в указанный регистр.

Описание команды:
Команда заносит в регистр устройства (процессора) с названием name значение data (обычный числовой формат, не hexdata)

Пример использования:

```
#...
WriteReg $DEV PC 0x1234
```

Данный скрипт заносит в регистр PC устройства $DEV число 0x1234.

5.2.9. ***ReadReg***

Формат команды:
`ReadReg devID name`

Команда возвращает считанное из регистра устройства с именем name данное в обычном числовом формате (Int, не hexdata).

Параметры:
`devID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`name` – имя регистра устройства devID

Описание команды:
Команда возвращает считанное из регистра устройства с именем name данное в обычном числовом формате (Int, не hexdata).

Пример использования:

```
#...
puts [ReadReg $DEV SP]
```

Данный скрипт выводит в консоль значение из регистра SP устройства $DEV.

*5.2.10.**WaitForBP***

Формат команды:
`WaitForBP devID timeout`

Команда не возвращает данные.

Параметры:
`devID` – Идентификатор устройства, полученный от команды **`xds add`**, к которому было произведено подключение по команде xds connect.
`timeout` – таймаут в 100-миллисекундных интервалах

Описание команды:
Команда ожидает останова устройства по точке останова, в цикле исполняя команду **`xds status`** и анализируя результат. В случае таймаута выходит в ОС с фатальной ошибкой.

Пример использования:

```
#...
WaitForBP $DEV 10
```

Данный скрипт ожидает останова устройства $DEV в течение секунды. Если останов произошел, выполнение скрипта продолжается. Иначе происходит выход с фатальной ошибкой таймаута.