Поддержка скриптов в SignalTap II

Вы можете запускать процедуры и делать настройки, описанные в этой главе, с помощью Tcl скриптов. Также вы можете запускать некоторые процедуры из командной строки. За подробной информацией об опциях команд скриптирования обратитесь к браузеру помощи Командной строки Quartus II и Tcl API. Для запуска браузера помощи наберите следующую команду в командной строке:

```
quartus_sh --qhelp ↵
```

Справочное руководство скриптирования Quartus II содержит такую же информацию в формате PDF.

За дополнительной информацией о Tcl скриптировании, обратитесь к главе «Tcl скриптирование» в томе 2 Настольной книги Quartus II.

За дополнительной информацией о скриптировании в командной строке, обратитесь к главе «Скриптирование в командной строке» в томе 2 Настольной книги Quartus II.

Опции командной строки SignalTap II

Для компиляции вашего проекта со встроенным логическим анализатором SignalTap II через командную строку, используйте команду `quartus_stp`. В таблице 14-13 показаны опции, которые помогут лучше понять, как лучше использовать выполнение `quartus_stp`.

Таблица 14-13. Опции командной строки SignalTap II

Опция	Использование	Описание
stp_file	quartus_stp --stp_file <stp_filename>	Назначение определяет **.stp** файл USE_SIGNALTAP_FILE в **.qsf** файле
enable	quartus_stp --enable	Создаёт назначения определённому **.stp** файлу в **.qsf** файле и изменяет ENABLE_SIGNALTAP в ON. Встроенный логический анализатор SignalTap II будет включен в ваш проект сразу после компиляции. Если **.stp** файл не определён в **.qsf** файле, можно использовать опцию --stp_file. Если опция --stp_file пропущена, используется текущее значение ENABLE_SIGNALTAP в **.qsf** файле.
disable	quartus_stp --disable	Удаляет связи **.stp** файла в **.qsf** файле и изменяет ENABLE_SIGNALTAP в OFF. Встроенный логический анализатор SignalTap II будет удалён из базы данных проекта сразу после компиляции. Если опция --disable пропущена, используется текущее значение ENABLE_SIGNALTAP в **.qsf** файле.
create_signaltap_hdl_file	quartus_stp -- create_signaltap_hdl_file	Создаёт **.stp** файл описания элемента SignalTap II в проекте, генерируемый мегафункцией встроенного логического анализатора SignalTap II с помощью менеджера плагинов MegaWizard. Вы должны использовать опцию --stp_file для правильного создания **.stp** файла. Аналогично команде **Создать файл SignalTap II из элементов проекта** в программе Quartus II.

В примере 14-6 показано, как компилируется проект с использованием встроенного логического анализатора SignalTap II из командной строки.

Example 14–6.

```
quartus_stp filtref --stp_file stp1.stp --enable ↵
quartus_map filtref --source=filtref.bdf --family=CYCLONE ↵
quartus_fit filtref --part=EP1C12Q240C6 --fmax=80MHz --tsu=8ns ↵
quartus_tan filtref ↵
quartus_asm filtref ↵
```

Команда quartus_stp --stp_file stp1.stp --enable создаёт переменную QSF и инструктирует программу Quartus II компилировать **stp1.stp** файл вместе с вашим проектом. Опция --enable должна применяться для правильной компиляции встроенного логического анализатора SignalTap II в вашем проекте.

В примере 14-7 показано, как создавать **.stp** файл после создания элемента встроенного логического анализатора SignalTap II с помощью менеджера плагинов MegaWizard.

Example 14–7.

```
quartus_stp filtref --create_signaltap_hdl_file --stp_file stp1.stp ↵
```

За дополнительной информацией о других операторах и опциях в командной строке, обратитесь к главе «Скриптирование в командной строке» в томе 2 Настольной книги Quartus II.

Опции Tcl команд SignalTap II

Оператор **quartus_stp** поддерживает интерфейс Tcl, который позволяет вам захватывать данные без запуска графической оболочки Quartus II. Вы не сможете запускать команды SignalTap II Tcl из внутренней консоли Tcl в GUI. Поэтому вы должны запускать их из командной строки с оператором **quartus_stp**. Для запуска Tcl файла, в котором находятся команды SignalTap II Tcl, используйте следующую команду:

```
quartus_stp -t <Tcl file> ↵
```

В таблице 14-14 показаны Tcl команды, которые вы можете использовать со встроенным логическим анализатором SignalTap II.

Таблица 14-14. Команды SignalTap II Tcl

Команда	Аргумент	Описание
`open_session`	`-name <stp_filename>`	Открывает определённый **.stp** файл. Все захваченные данные хранятся в этом файле.
`run`	`-instance <instance_name>` `-signal_set <signal_set> (optional)` `-trigger <trigger_name> (optional)` `-data_log <data_log_name> (optional)` `-timeout <seconds> (optional)`	Запускает анализатор. Эта команда должна иметь все нужные аргументы для правильного старта анализатора. Вы можете опционально определить имя журнала данных, который вы хотите создать. Если состояние триггера не произошло, вы можете выбрать величину таймаута до остановки анализатора.
`run_multiple_start`	нет	Определяет старт набора команд `run`. Используйте эту команду для одновременного старта различных элементов захвата данных. Добавьте эту команду перед набором команд запуска, определяющих захват данных. Вы должны использовать эту команду вместе с командой `run_multiple_end`. Если команда `run_multiple_end` отсутствует, команда `run` не выполняется.
`run_multiple_end`	нет	Определяет окончание набора команд `run`. Используйте эту команду для одновременного старта различных элементов захвата данных. Добавьте эту команду после набора команд запуска, определяющих захват данных.
`stop`	нет	Останавливает захват данных.
`close_session`	нет	Закрывает текущий открытый **.stp** файл. Вы не можете запустить анализатор после того, как **.stp** файл закрыт.

За дополнительной информацией о командах SignalTap II Tcl, обратитесь к помощи Quartus II.

Пример 14-8 состоит из скрипта, который использует непрерывный захват данных. После срабатывания триггера, данные захватываются и сохраняются в журнале данных.

Example 14–8.

```
#opens signaltap session
open_session -name stp1.stp
#start acquisition of instance auto_signaltap_0 and
#auto_signaltap_1 at the same time
#calling run_multiple_end will start all instances
#run after run_multiple_start call
run_multiple_start
run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger /
trigger_1 -data_log log_1 -timeout 5
run -instance auto_signaltap_1 -signal_set signal_set_1 -trigger /
trigger_1 -data_log log_1 -timeout 5
run_multiple_end
#close signaltap session
close_session
```

Когда скрипт закончится, откройте **.stp** файл, который вы используете для захвата данных, чтобы посмотреть содержимое журнала данных.

Пример проекта: использование встроенного логического анализатора SignalTap II в системах разработки SOPC

Система в этом примере состоит из нескольких компонентов, включая процессор Nios, контроллер памяти прямого доступа (DMA), память в чипе и интерфейс с внешней SDRAM памятью. В этом примере, процессор Nios исполняет простую программу на C из внутри чиповой памяти и ожидает нажатия на кнопку. После нажатия кнопки, процессор инициирует передачу DMA, которую вы анализируете с использованием встроенного логического анализатора SignalTap II.

За дополнительной информацией об этом примере и использовании встроенного логического анализатора SignalTap II в системах разработки SOPC, обратитесь к AN 323: Использование встроенного логического анализатора SignalTap II в системах разработки SOPC и к AN 446: Отладка Nios II систем встроенным логическим анализатором SignalTap II.

Примеры приложений процесса настраиваемого триггера

Процесс настраиваемого триггера во встроенном логическом анализаторе SignalTap II полностью подходит для организации состояний триггера и для точного контроля за буфером захвата. В этой секции содержатся два примера приложений для определения процесса настраиваемого триггера во встроенном логическом анализаторе SignalTap II. Оба примера запросто копируются и вставляются в окно описания конечного автомата, используя режим отображения экрана состояний **Все состояния в одном окне**.

За дополнительными примерами процессов триггера, обратитесь к странице «Примеры отладки проектов в чипе в Quartus II».

Пример проекта 1: Определение позиции настраиваемого триггера

Действия буфера захвата могут быть связаны с опциональным аргументом пост-счёта. Этот аргумент пост-счёта позволяет вам определять позиции настраиваемого триггера для каждого сегмента буфера захвата. В примере 14-9 показано, как применяются позиции триггера ко всем сегментам буфера захвата. В этом примере описан процесс триггера для буфера захвата, разделённого на четыре сегмента. Если каждый сегмент имеет глубину 64 отсчёта, то позиция триггера для каждого буфера будет на 34 отсчёте. Захват останавливается после того, как заполнятся все четыре сегмента.

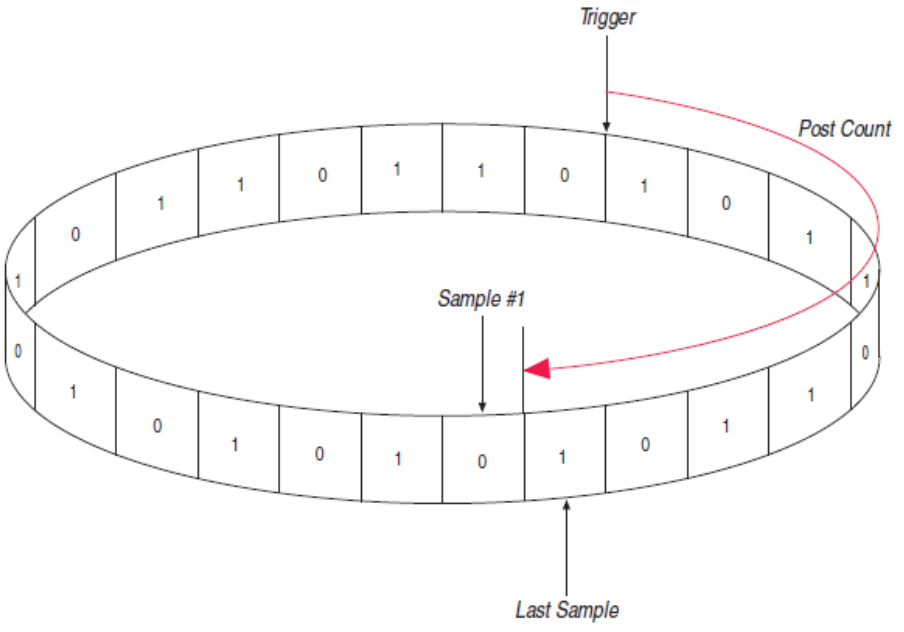
Example 14–9.

```
if (c1 == 3 && condition1)
    trigger 30;
else if ( condition1 )
begin
    segment_trigger 30;
    increment c1;

end
```

Каждый сегмент работает как несегментный буфер, который непрерывно обновляет содержимое памяти значением сигнала. Последний захват прежде чем остановить буфер, отображается на вкладке **Данные** как последний номер отсчёта в действующем сегменте. Позиция триггера в действующем сегменте определяется как *N – заполнение пост-счета*, где N – это количество отсчётов на сегмент. На рисунке 14-55 показана позиция триггера.

Figure 14–55. Specifying a Custom Trigger Position



Пример проекта 2: Захват, когда срабатывает triggercond1 ожидать десять тактов между triggercond2 и triggercond3

Описание процесса настраиваемого триггера прекрасно подходит для подсчёта последовательности событий, прежде чем защёлкнуть буфер захвата. В примере 14-10 показан процесс отсчётов. Этот пример использует три базовых состояния триггера, конфигурируемые на вкладке **Установки** в SignalTap II.

В этом примере буфер захвата защёлкивается, когда состояние `condition1` наступает после `condition3` и происходит десять раз прежде, чем происходит состояние `condition3`. Если `condition3` происходит прежде, чем десять раз `condition1`, конечный автомат переходит в перманентное состояние ожидания.

Example 14–10.

```
state ST1:

if ( condition2  )
begin
   reset c1;
   goto ST2;
end


State ST2 :
if ( condition1 )
    increment c1;

else if (condition3 && c1 < 10)
    goto ST3;

else if ( condition3 && c1 >= 10)
    trigger;

ST3:
goto ST3;
```

Заключение

Индустрия FPGA продолжает создавать технологические улучшения, устаревшие технологии должны заменяться новыми, которые увеличивают продуктивность. Встроенный логический анализатор обладает преимуществами традиционных логических анализаторов, за исключением некоторых недостатков элементов тестового оборудования. В этой версии встроенного логического анализатора содержатся некоторые новые и обновлённые средства, позволяющие вам захватывать и анализировать внутренние сигналы в вашем FPGA, позволяя вам находить источник проблемы в проекте за максимально быстрое время.