10. SOPC Builder Component Development Walkthrough

10. Сквозной контроль разработки компонента SOPC Builder

В этой главе описываются составные части собственного компонента SOPC Builder, и приводится руководство по процессу создания примера собственного компонента, его интеграции в систему и аппаратному тестированию.

Эта глава состоит из следующих секций:

- "Процесс разработки компонента" на стр. 10-2
- "Пример проекта: аппаратный ускоритель функции контрольной суммы" на стр. 10-4. В этом примере показывается, как разрабатывать компонент с двумя режимами по шине Avalon® с распределением в памяти (Avalon-MM): мастер и слейв.
- "Разделяемые компоненты" на стр. 10-7. В этой секции показано, как вы сможете использовать компоненты в других системах, или разделить их с другими разработчиками.
- "Файлы информации о системе (.sopcinfo)" на стр. 10-7.

Компоненты SOPC Builder и редактор компонентов

Компонент SOPC Builder обычно состоит из файлов четырёх типов:

- Файлы HDL – определяют аппаратную функциональность компонента
- Аппаратный файл описания компонента (**_hw.tcl**) – описывает соответствующие характеристики SOPC Builder, такие как работа интерфейса. Этот файл создаётся редактором компонента.
- Файлы на языке Си – определяют карту регистров компонента и программного драйвера, чтобы позволить программам контролировать компонент.
- Программный файл описания компонента (**_sw.tcl**) – используется инструментом создания программы для использования и компиляции кода драйвера компонента.

Редактор компонента руководит процессом создания вашего компонента. Вы можете инсталлировать этот компонент в систему SOPC Builder и создать соединения тем же способом, что и для других компонентов SOPC Builder. Вы также можете сделать доступным свой компонент для других разработчиков.

За информацией о создании **_sw.tcl** файла смотрите главу "Разработка драйверов устройств для слоя аппаратной абстракции" в настольной книге программиста Nios II.

Необходимое условие

Материал этой главы подразумевает, что вы знакомы с:

- созданием систем SOPC Builder. Подробнее в главе 1, "Введение в SOPC Builder".
- компонентами SOPC Builder. Подробнее в главе 4, "Компоненты SOPC Builder"
- основами интерфейса Avalon-MM.

Аппаратные и программные требования

Чтобы использовать примеры проекта в этой главе, в дополнении к текущей версии программы Quartus II и Nios II Embedded Design Suite, вы должны иметь следующее:

- Файлы проекта для примера проекта – Гиперссылка на файлы проекта находится рядом с этим руководством пользователя на странице литературы по SOPC Builder.
- Плату разработке Nios II и загрузочный кабель Altera® USB-Blaster™ – вы можете использовать одну из следующих плат разработки Nios II:
  - Stratix® II Edition, RoHS compliant version
  - Cyclone® II Edition

Если у вас нет платы разработки, вы можете просто следовать пунктам разработки устройства. Конечно, вы не сможете загрузить законченную систему без рабочей платы, но сможете просимулировать эту систему.

Вы можете бесплатно загрузить программу Quartus II Web Edition и Nios II EDS, Evaluation Edition со страницы центра загрузки Altera на www.altera.com.

Процесс разработки компонента

В этой секции дано общее представление о процессе разработки компонентов для SOPC Builder.

Обычные пункты проектирования

Обычная последовательность проектирования компонента для SOPC Builder содержит следующие элементы:

1. Спецификация и определения
   a. Определение функциональной схемы компонента
   b. Определение интерфейсов компонента, таких как Avalon с распределением в памяти (Avalon-MM), потоковый Avalon (Avalon-ST), прерываний и прочих интерфейсов.
   c. Определение требований по тактированию компонента; то есть, какой интерфейс синхронен какому входу тактового сигнала.
   d. Если вы хотите, чтобы микропроцессор контролировал компонент, определите его интерфейс с программой, такой как карта регистров.
2. Реализуйте компонент на языке VHDL или Verilog HDL.

3. Импортируйте компонент в SOPC Builder.
   a. Используйте редактор компонента для создания **_hw.tcl** файла описания компонента.
   b. Инсталлируйте компонента в систему SOPC Builder.

Во время импорта HDL файла с использованием редактора компонента, некоторые параметры, чьё определение зависит от определения других параметров, вызывают ошибку.

В примере 10-1 показано декларирование параметра DEPTH, являющееся синтаксически корректным для языка Verilog HDL в программе Quartus II, но вызывающее ошибку при проверке синтаксиса в редакторе компонентов.

Example 10–1. DEPTH Parameter

```
parameter WIDTH = 32;
parameter DEPTH = ((WIDTH == 32) ? 8 : 16);
```

Чтобы избежать этой ошибки, используйте localparam вместо parameter, как показано в примере 10-2.

Example 10–2. localparam Parameter

```
parameter WIDTH = 32;
localparam DEPTH = ((WIDTH == 32)?8:16);
```

SOPC Builder поддерживает только следующие типы портов VHDL std_logic и std_logic_vector.

4. Разработайте программный драйвер, который можно делать параллельно с аппаратной реализацией. Создайте драйвер компонента, содержащий заголовочный файл Си, в котором определена для программы карта регистров на аппаратном уровне.
   За подробной информацией обратитесь к настольной книге программиста Nios II.
5. Выполните тестирование системы, например:
   a. Протестируйте доступ на уровне регистров к компоненту в устройстве или просимулируйте его, используя микропроцессор, например Nios II.
   b. Выполните установку контрольных точек.

Аппаратное проектирование

Как любой процесс логического проектирования, разработка аппаратного компонента SOPC Builder начинается после стадии спецификации. Создание HDL проекта – зачастую итеративный процесс, при котором вы пишете и сверяете HDL логику с заданной спецификацией.

Архитектура обычного компонента состоит из следующих функциональных блоков:

- Логика задачи – реализует основную функцию компонента. Логика задачи зависит от проекта.
- Логика интерфейса – предлагает стандартный способ передачи данных к или получения данных для компонентов, и контролирования функционирования компонентов.

За подробной информацией обратитесь к главе "Спецификация интерфейса Avalon".

На рис. 10-1 показаны блоки верхнего уровня компонента контрольной суммы, которые имеют оба интерфейса Avalon-MM мастер и слейв.

Процесс создания аппаратной части SOPC Builder, включая то, как выбрать и реализовать карту регистров, описан в главе "Использование инструмента разработки программы Nios II" в настольной книге программиста Nios II. Руководство по написанию драйвера устройства описаны в главе "Разработка драйвера устройства для слоя аппаратной абстракции" в настольной книге программиста Nios II.

Пример проекта: аппаратный ускоритель функции контрольной суммы

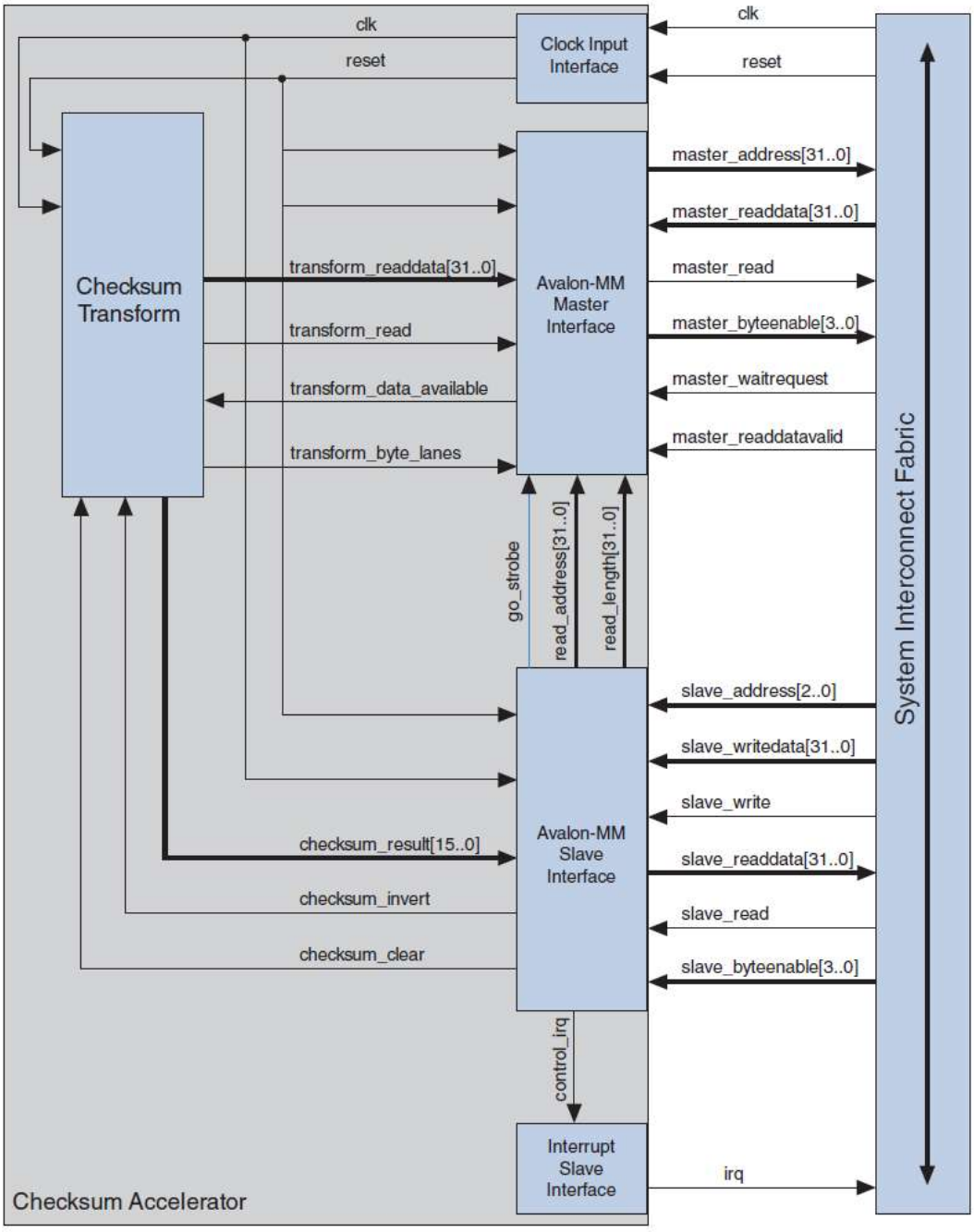
Altera предлагает пример проекта аппаратного ускорителя функции контрольной суммы для демонстрирования пунктов создания компонента и инсталляции его в систему. Этот пример проекта доступен для загрузки со страницы Литература на веб-сайте Altera. В файле **readme.pdf**, включенном в архив загрузочного файла, описано, как создавать и компилировать аппаратный проект, а также описано, как использовать аппаратный ускоритель функции контрольной суммы в вашем проекте.

Вы можете использовать алгоритм контрольной суммы в сетевых приложениях, в которых целостность данных должна контролироваться приёмным устройством. Алгоритм контрольной суммы аккумулирует данные по методу суммации "циклического переноса", которая работает так, бит переноса из аккумулятора складывается с самым младшим битом следующего числа. После того, как данные аккумулированы, вы можете использовать результат для проверки целостности данных в буфере данных. Поскольку алгоритм контрольной суммы работает над буфером данных, вы можете реализовать его более эффективно в виде мастера конвейерного чтения. Мастер конвейерного чтения непрерывно регистрирует транзакции чтения, минимизировав эффект задержки чтения памяти. Ускоритель функции контрольной суммы может читать данные и подсчитывать результат контрольной суммы каждый тактовый цикл, так вы не сможете сделать с процессором общего назначения.

Для аппаратного ускорителя функции контрольной суммы необходима информация от хост процессора, такая как базовый адрес буфера, длина буфера и варианты контрольных сигналов. В результате, аппаратный ускоритель размещается в Avalon-MM слейв интерфейсе, чтобы хост процессор смог контролировать операции мастера конвейерного чтения. Хост процессор также получает доступ к результатам контрольной суммы через слейв интерфейс. Каждый фрагмент информации, посылаемый или принимаемый хост процессором, доступен отдельно в регистровом файле, реализованном в слейв интерфейсе. Например, сигналы статуса и контроля реализованы в отдельных регистрах, поскольку они содержат информацию различного значения, и имеют различные функции доступа.

Аппаратные ускорители могут работать параллельно с хост процессором; поэтому добавление интерфейса посылки прерывания в аппаратный ускоритель улучшает характеристики системы. Пока ускоритель работает с буфером, хост процессор может выполнять другие задачи, такие как подготовка другого буфера для передачи. Прерывание возникает после подсчёта контрольной суммы. Хост процессор прерывается аппаратным ускорителем, извещающим его, что получен результат контрольной суммы. Хост процессор может прочитать значение контрольной суммы и сбросить прерывание записью в регистр статуса через слейв интерфейс ускорителя.

Figure 10–1. Checksum Component with Avalon-MM Master and Slaves



Программное проектирование

Если вы хотите использовать микропроцессор для контроля над вашим компонентом, вы должны предоставить программные файлы, в которых отображено программное описание компонента. Как минимум, вы должны определить карту регистров для каждого слейва на Avalon-MM, который доступен процессору.

В примере проекта контрольной суммы, вы можете увидеть пример программы драйвера в директории *<projectdir>***/ip/checksum_accelerator**, которая находится в корневой папке hardware и software собственного блока контрольной суммы.

Программный драйвер абстрагирует от аппаратного уровня компонента, так чтобы программа имела доступ к компоненту в верхнем уровне. Функция драйвера – предоставлять API программе доступ к устройству. Программные требования варьируются в зависимости от задач компонента. Обычно – это процедуры инициализации устройства, чтения и записи данных.

Когда вы разрабатываете программный драйвер, вы можете посмотреть программные файлы для готовых компонентов. Инсталлятор IP предлагает несколько компонентов, которые вы можете использовать для справки. Их примеры можно найти в директории **<***Nios II EDS install path***>/components/**.

За подробной информацией о написании драйверов для слоя аппаратной абстракции Nios II (HAL), обратитесь к главе "Разработка драйвера устройства для слоя аппаратной абстракции" в настольной книге программиста Nios II.

Верификация компонента

Вы можете проверять компонент на стадиях инкремента, когда вы закончите большую часть проекта. Сначала нужно проверить аппаратную логику как модуль (который должен состоять из нескольких малых частей верификации), а затем верифицировать компонент в системе.

Системная консоль

Системная консоль – это интерактивная TCL консоль, доступная в SOPC Builder, которая предоставляет вам доступ к чтению и записи при отладке функций, доступных в вашей FPGA логике. Вы можете использовать системную консоль для контролирования и запроса состояний процессора Nios II, результатов транзакций по шине Avalon, обмену с платой и доступа через узлы JTAG UART или отладчика системного уровня (SLD).

За дополнительной информацией обратитесь к "Руководству пользователя по системной консоли".

Верификация на системном уровне

После того, как вы создадите **_hw.tcl** файл в редакторе компонента, вы можете инсталлировать компонент в систему и проверить его функционирование в составе всей системы SOPC Builder.

SOPC Builder предлагает поддержку верификации на системном уровне для HDL симуляторов, таких как as ModelSim®. SOPC Builder автоматически создаёт файл тестового стенда (test bench) для верификации на системном уровне.

Вы можете включить в вашу систему процессор Nios II, чтобы улучшить функциональность симуляции на стадии верификации. Даже если ваш компонент никак не связан с процессором Nios II, автоматически сгенерированная среда симуляции ModelSim предлагает простую в использовании отправную точку.

Разделяемые компоненты

Когда вы создадите компонент, редактор компонента сохранит **_hw.tcl** файл в той же директории, что и HDL файл верхнего уровня. В некоторых местах, файлы, на которые ссылается **_hw.tcl** файл, могут быть заданны в себе относительно **_hw.tcl** файла, такие файлы могут быть запросто перемещены и скопированы. Чтобы сделать компонент разделяемым (совместное использование компонента), включите его в библиотеку IP.

За дополнительной информацией обратитесь к секции "Поиск пути компонента SOPC Builder" в главе 4, "Компоненты SOPC Builder"

Файлы информации о системе (.sopcinfo)

Каждый раз при генерировании системы SOPC Builder, также генерируется файл <*mysystem*>**.sopcinfo**, содержащий следующую информацию:

- Проект SOPC Builder, включая:
  - Имя и версию инструментария
  - Язык HDL
- Каждый модуль, инсталлированный в системе, включая:
  - Имя и версию
  - Какую-либо информацию об интерфейсе, найденную на диске, например, имена сигналов и их тип, свойства интерфейса, размещение тактовых доменов.
  - Имена и значения параметров
- Каждое соединение, включая:
  - Соединения между компонентом и интерфейсом
  - Базовый адрес, интерфейсы Avalon-MM, число интерфейсов IRQ
  - Схема памяти, как она видится каждым мастером системы

Файл **.sopcinfo** – это только файл отчёта, он не может быть отредактирован в SOPC Builder.

Оглавление