Создание пользовательского IP-ядра в Qsys/Platform Designer

Автор: **andrewkushchenko**

Рецензенты: **Intekus**

KeisN13

2019 г.

Оглавление

Аннотация

В статье рассмотрен способ создания пользовательских IP-ядер в Intel Qsys/Intel Platform Designer. И Intel Qsys, и Intel Platform Designer являются компонентами среды Intel Quartus. Разница между Qsys и Platform Designer невелика: во втором незначительно расширен функционал. Qsys используется в более старых версиях Quartus, более новые – поставляются с Platform Designer. У меня на машине установлен Quartus Prime Lite 17.1, который идет уже с Platform Designer, так что именно он и будет использован для примера.

Введение

Концепция добавления пользовательских IP-ядер в Platform Designer проста: описать все интерфейсы модуля. Это необходимо для правильного подключения различных модулей друг к другу. Platform Designer поддерживает достаточно широкий спектр интерфейсов, которые можно видеть на рисунке 1.

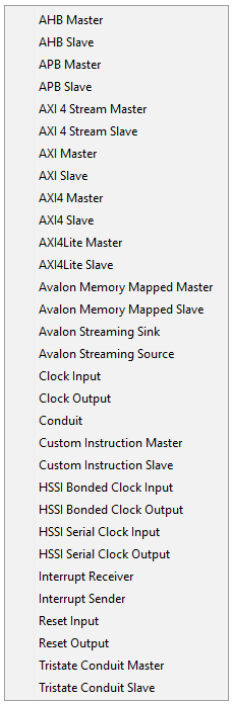

Рисунок 1 – Список поддерживаемых интерфейсов

Для примера возьмем простой модуль, написанный на VHDL, имеющий 3 интерфейса: Avalon-STin, Avalon-STout и Avalon-MMcsr. Модуль должен принимать поток данных по интерфейсу Avalon-STin, выполнять над данными некоторые преобразования (это сейчас для нас не важно) и выдавать результат по интерфейсу Avalon-STout. А Avalon-MMcsr будет при этом задавать параметры работы модуля.

В листинге представлен код модуля

```
library ieee;
use ieee.std_logic_1164.all;

entity stream_converter is
    generic (
        CONST_A : integer := 16#00#;
        CONST_B : integer := 16#F5#
    );
    port (
        clk   : in std_logic;
        reset : in std_logic;

        stream_in_data  : in std_logic_vector(31 downto 0);
        stream_in_sop   : in std_logic;
        stream_in_eop   : in std_logic;
        stream_in_valid : in std_logic;
        stream_in_ready : out std_logic;

        stream_out_data  : out std_logic_vector(31 downto 0);
        stream_out_sop   : out std_logic;
        stream_out_eop   : out std_logic;
        stream_out_valid : out std_logic;
        stream_out_ready : in std_logic;

        cp_address      : in std_logic_vector(5 downto 0);
        cp_write        : in std_logic;
        cp_writedata    : in std_logic_vector(31 downto 0);
        cp_read         : in std_logic;
        cp_readdata     : out std_logic_vector(31 downto 0);
        cp_waitrequest  : out std_logic
    );
end entity;

architecture rtl of stream_converter is
begin
end architecture;
```

Создание IP-ядра

После того, как модуль создан, можно создавать IP-ядро в Platform Designer. Для этого откроем Platform Designer, и на панели IP Catalog нажмем кнопку New.

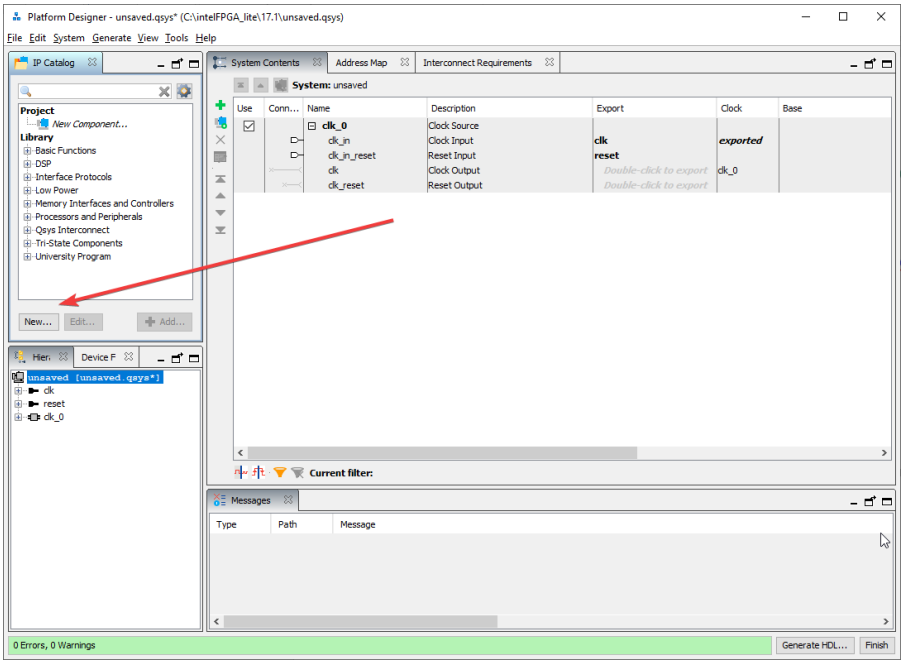

Рисунок 2 – Главное окно Platform Designer

После нажатия кнопки мы увидим окно создания нового IP-ядра

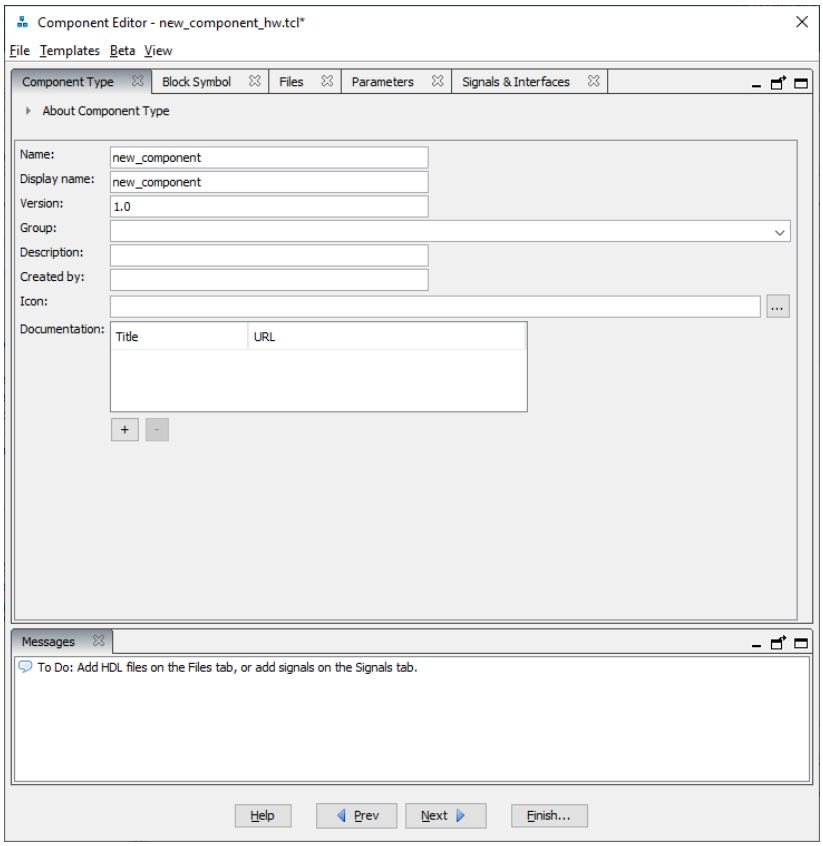

Рисунок 3 – Окно создания нового IP-ядра

На самой первой вкладке необходимо заполнить информацию об IP-ядре: имя, версию, группу в IP-каталоге и т. д. Мы зададим только поля Name и Display name. В большинстве случаев

этого будет достаточно. В оба поля напишем название модуля «Stream_converter». Имя может быть любым и не обязательно соответствовать названию модуля.

Далее перейдем на вкладку Files и добавим исходные файлы к IP-ядру.

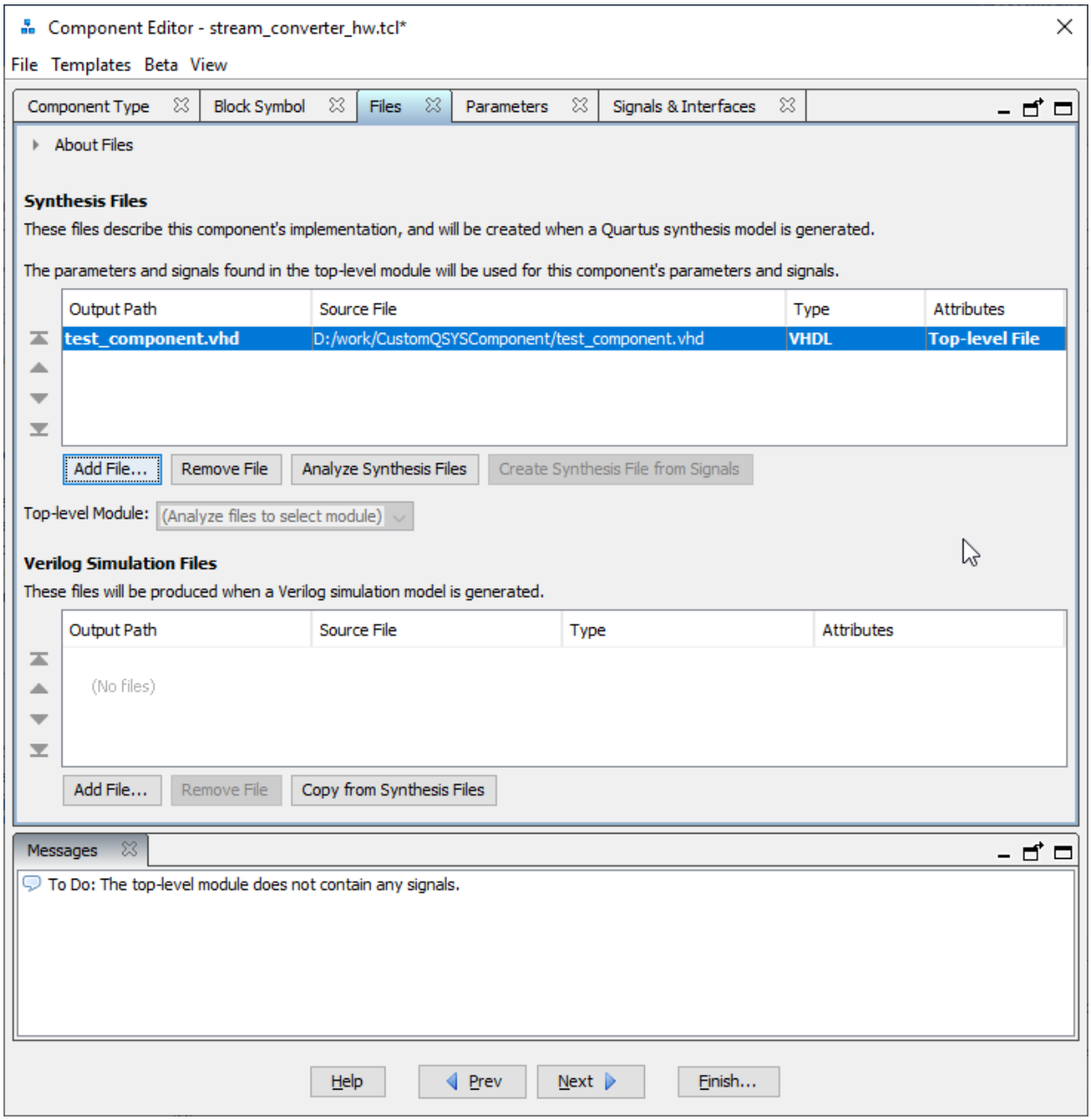

Рисунок 4 – Добавление исходных файлов к IP-ядру

Можно добавить все файлы, который входят в состав модуля, но я предпочитаю добавлять только обертку верхнего уровня модуля, т. к. все добавленные файлы будут копироваться в каталог сгенерированных файлов, что не всегда удобно.

После добавления всех необходимых файлов жмем кнопку Analyze Synthesis Files и ждем, пока Platform Designer проверит синтаксис добавленных файлов.

Теперь на вкладке Block Symbol мы можем уже увидеть все интерфейсы, которые Platform Designer смог определить автоматически.

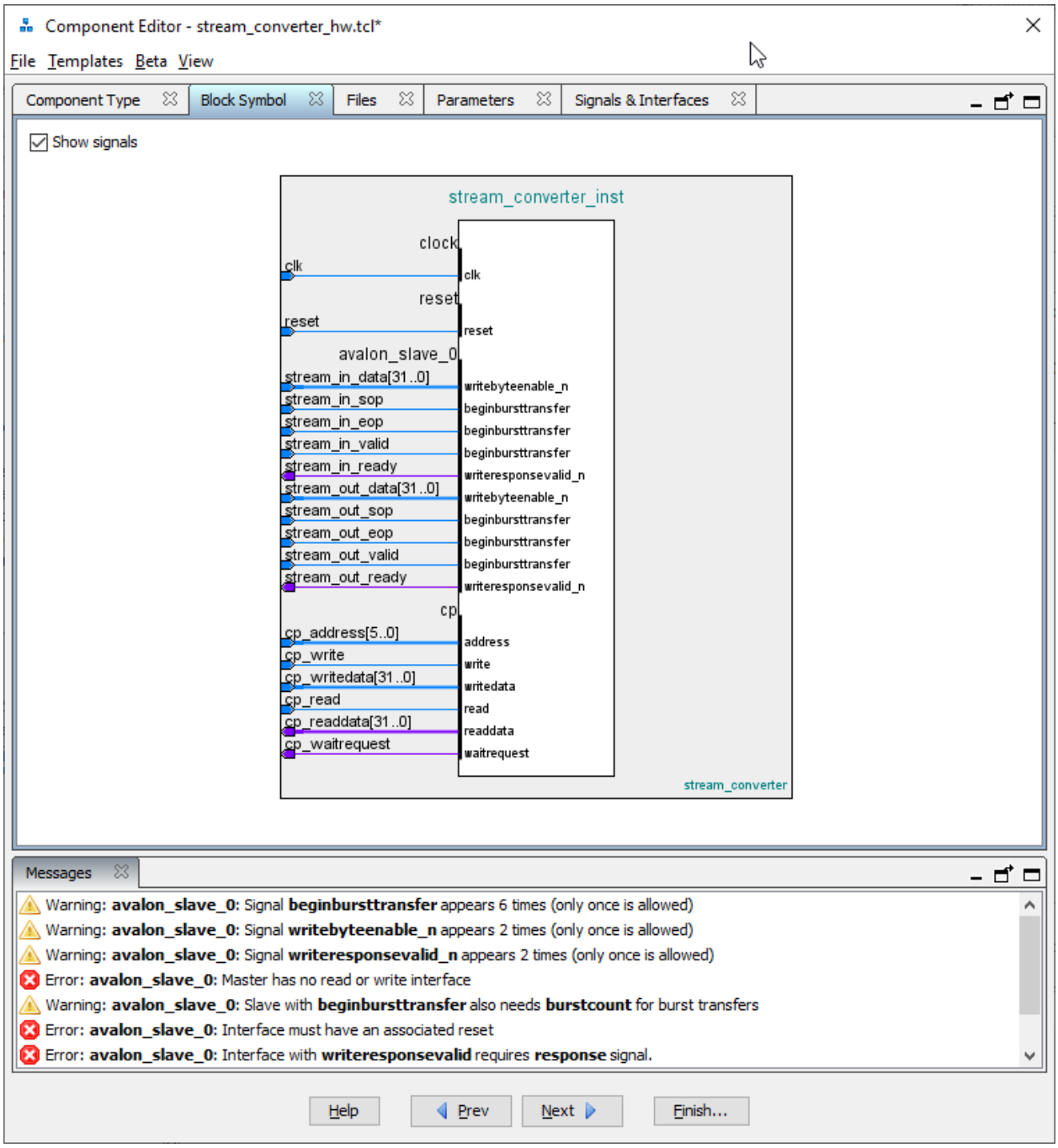

Рисунок 5 – Block Symbol

Как видим, Platform Designer корректно определил интерфейс Avalon-MM (ControlPort), а вот оба stream интерфейса определились неправильно. Теперь на вкладке Signals & Interfaces необходимо подправить некорректно определённые интерфейсы.

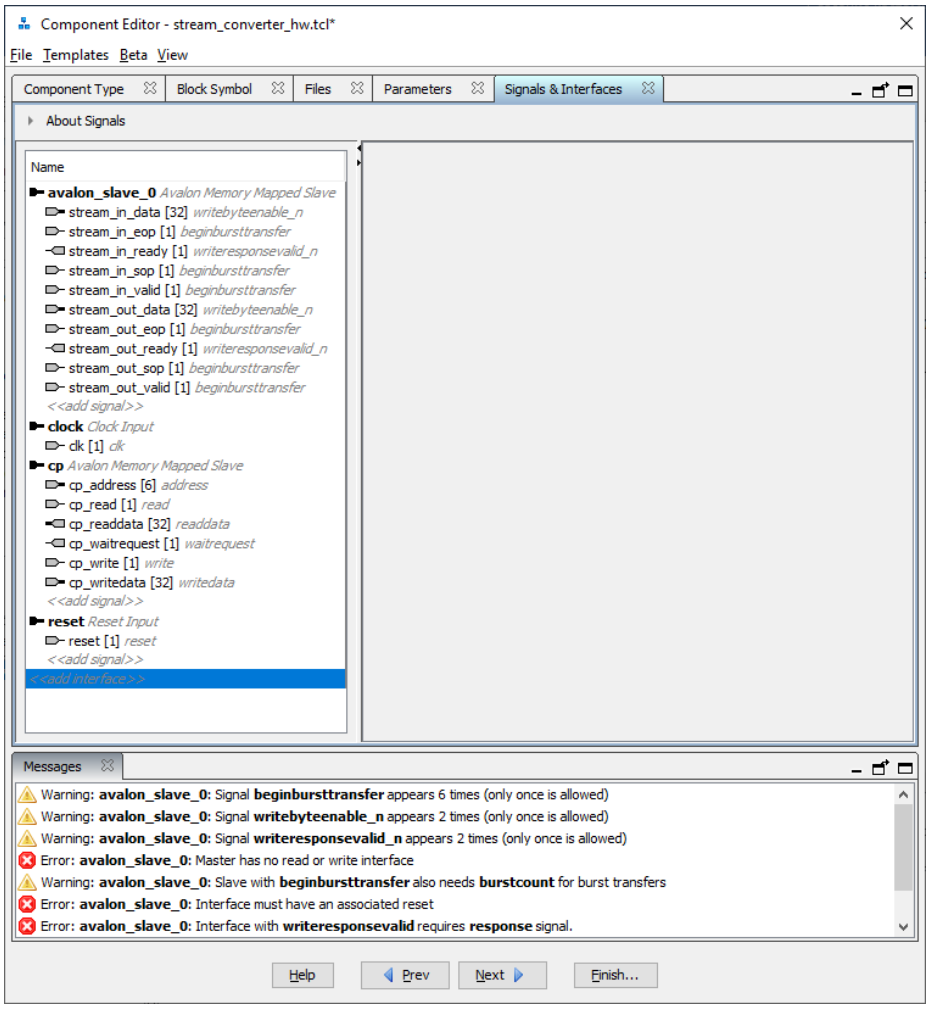

Рисунок 6 – Signals & Interfaces

Тут мы видим, что Platform Designer распознал 2 stream интерфейса, как 1 memory-mapped. Для исправления ситуации создадим 2 stream интерфейса вручную, нажав на кнопку <<addinterface>>, и перетянем туда соответствующие сигналы. Т.к. у нас один Avalon-ST интерфейс приемный, а второй передающий, то необходимо создать один интерфейс Avalon Streaming Sink, а второй Avalon Streaming Source. Пустой автосгенерированный интерфейс можно удалить.

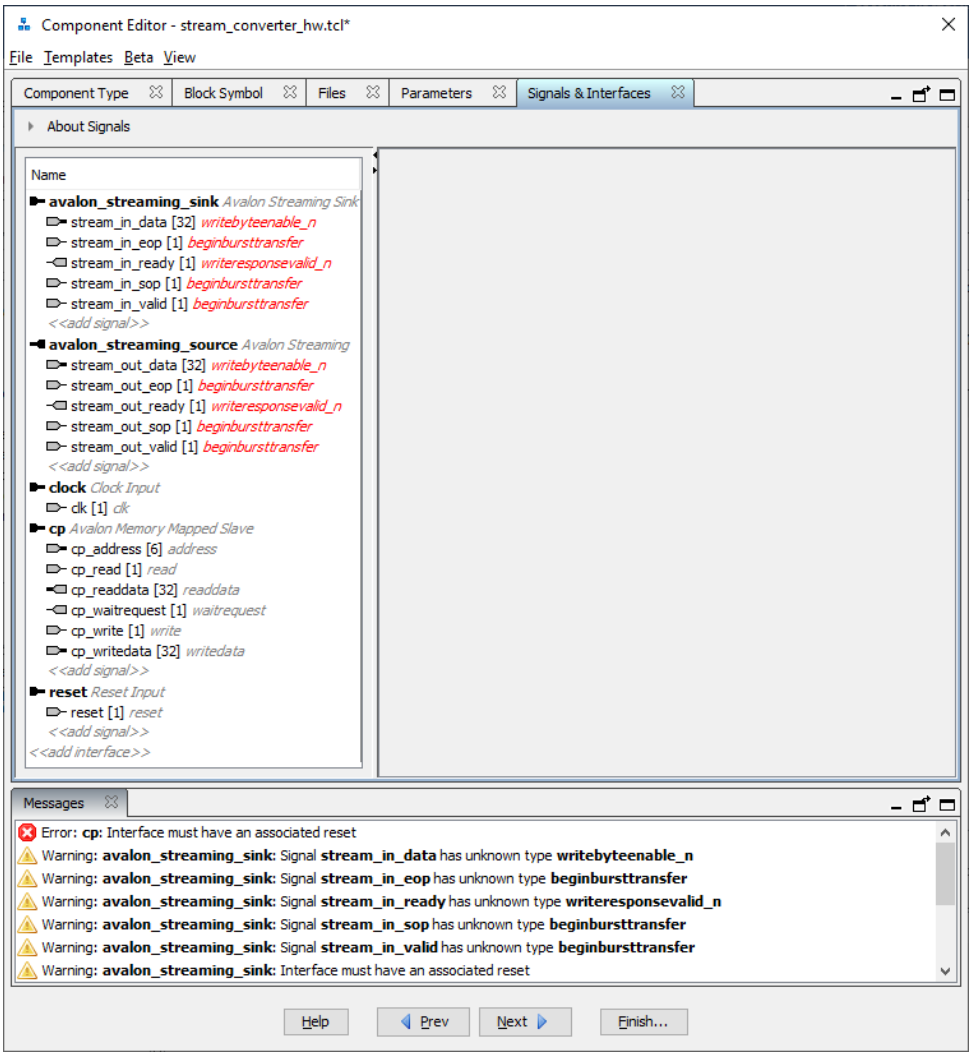

Рисунок 7 – Создание новых интерфейсов

Как видим, теперь у нас и правда 2 Avalon-ST интерфейса, но все сигналы помечены красным, и в логе будет множество ошибок, т. к. сигналы имеют неправильное назначение. В каждом интерфейсе необходимо задать правильные функции для всех сигналов, чтобы в Platform Designer корректно подключать между собой разные модули.

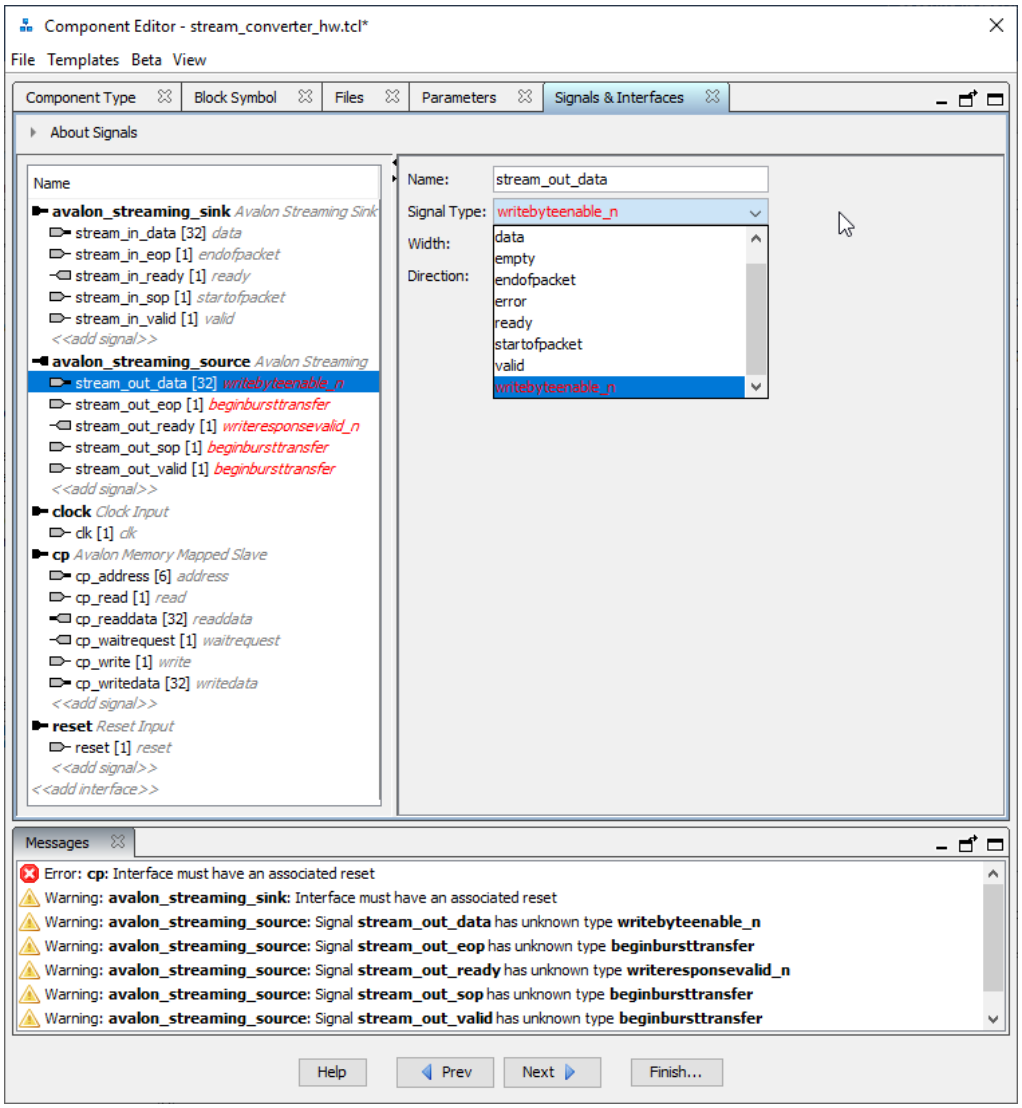

Рисунок 8 – Назначение функций сигналам

После назначения функций осталась ещё одна ошибка и пара предупреждений.

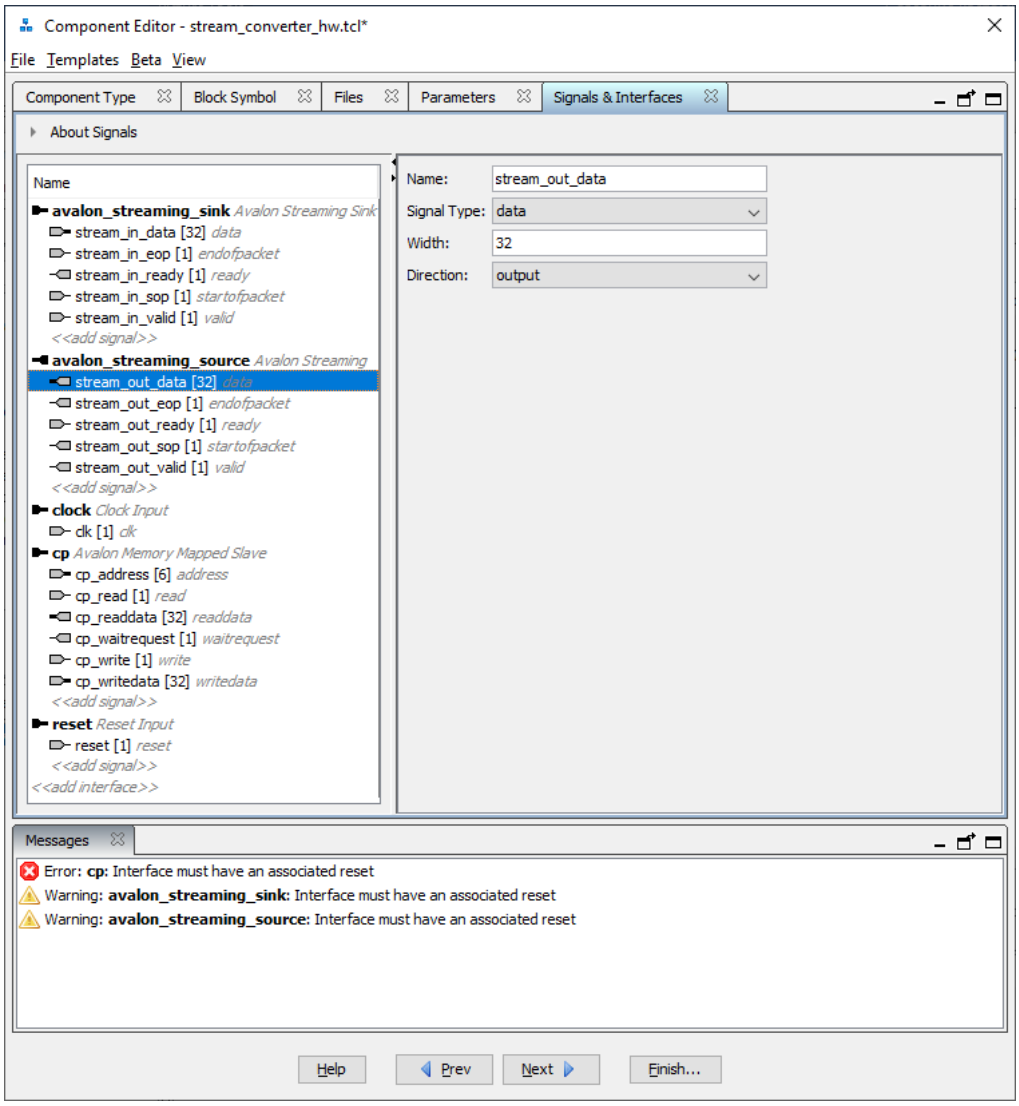

Рисунок 10 – Ошибки назначения сброса

Необходимо для каждого интерфейса типа Avalon-MM и Avalon-ST назначить сигнал сброса. Обязательным ещё является тактовый сигнал, но он определился автоматически. Задавать тактовый сигнал и сигнал сброса для интерфейсов необходимо для того, чтобы Platform Designer мог корректно подключать интерфейсы из разных клоковыми доменов. Например, наше IP-ядро может работать на системной частоте 100 МГц, но интерфейс программирования может быть подключен к PCIe контроллеру, который работает на частоте 125 МГц (его интерфейсная часть со стороны FPGA). В случае с доступом к регистрам модуля мы можем просто соединить наше IP-ядро и PCIe-контроллер, не думая о CDC (Clock Domain Crossing). Platform Designer автоматически вставит переход между частотами. Но это не очень правильный путь, т.к. подобные преобразования потребляют дополнительные ресурсы, и в большом проекте Platform Designer может автоматически поставить десяток переходников между клоковыми доменами и, например, преобразователей разрядности шины. Но оптимизация системы в Platform Designer – это отдельная обширная тема, её мы рассмотрим в другой раз.

После последних исправлений у нас больше нет никаких предупреждений, и остается только нажать кнопку Finish для завершения создания IP-ядра.

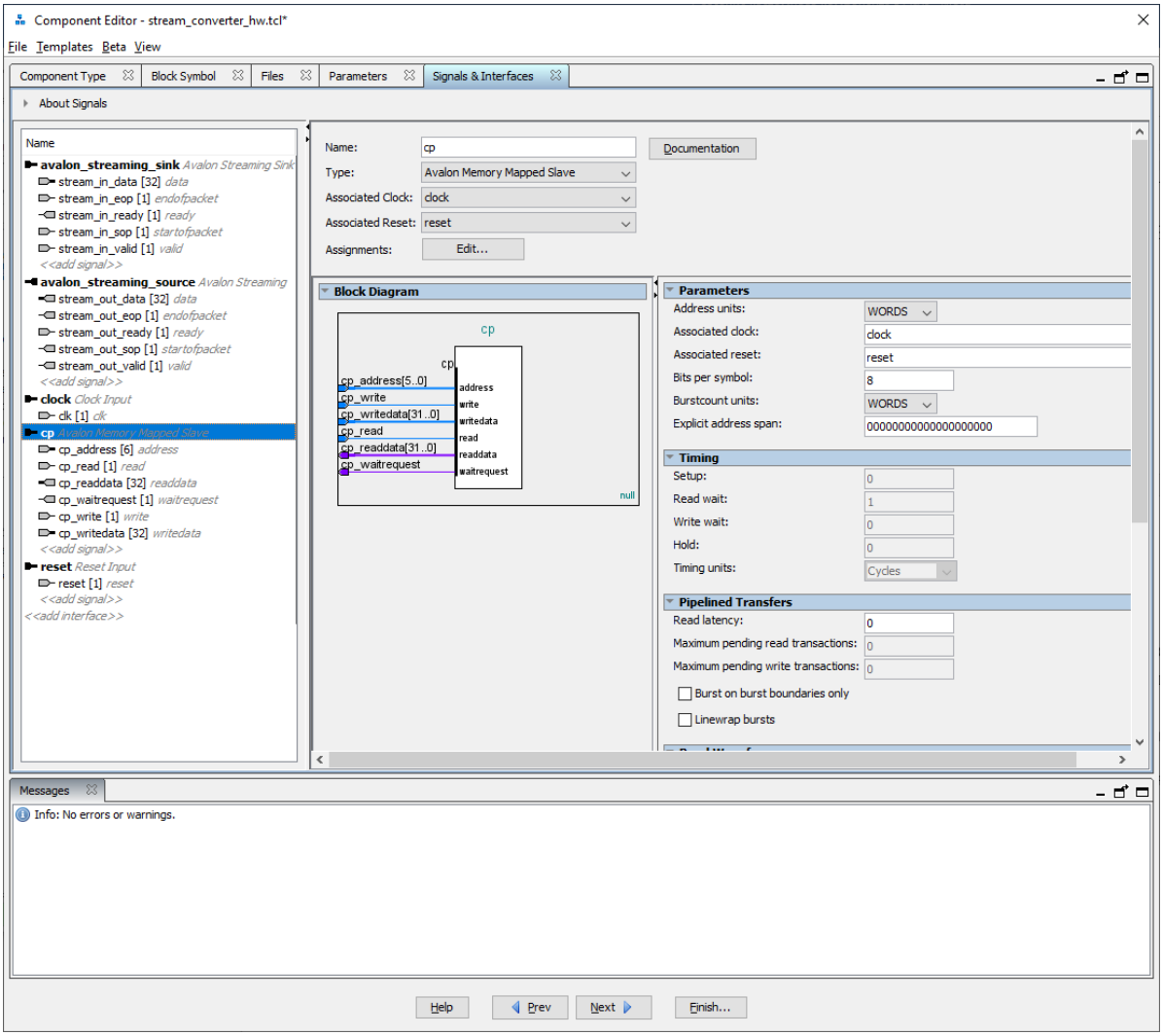

Рисунок 11–Детальная информация интерфейса Avalon-MM

На этом этапе дополнительно можно настроить параметры интерфейсов. В мануале на шину Avalon описано множество её параметров. Например, адресация словная или байтовая, задержка чтения и т. д. В большинстве случаев стандартных настроек достаточно, так что создание IP-ядра можно завершать.

Теперь в окне IP catalog у нас появился новое IP-ядро.

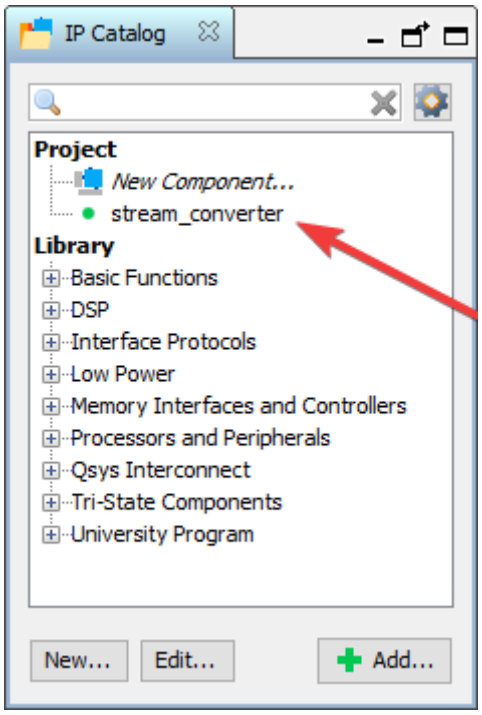

Рисунок 12 – Новый компонент в окне IP-catalog

Теперь его можно использовать в своей системе. Добавим в систему наше IP-ядро и ещё несколько библиотечных для демонстрации. Поставим JTAG to Avalon Master Bridge, чтобы иметь доступ к регистрам нашего модуля через JTAG и добавим Clocked Video Input/Output для имитации приема и передачи видео. Как видно, интерфейсы нашего модуля легко подключаются к интерфейсам IP-ядер от самого Intel. Таким образом, можно создать библиотеку из необходимого набора пользовательских IP-ядер и достаточно быстро собирать большие системы.

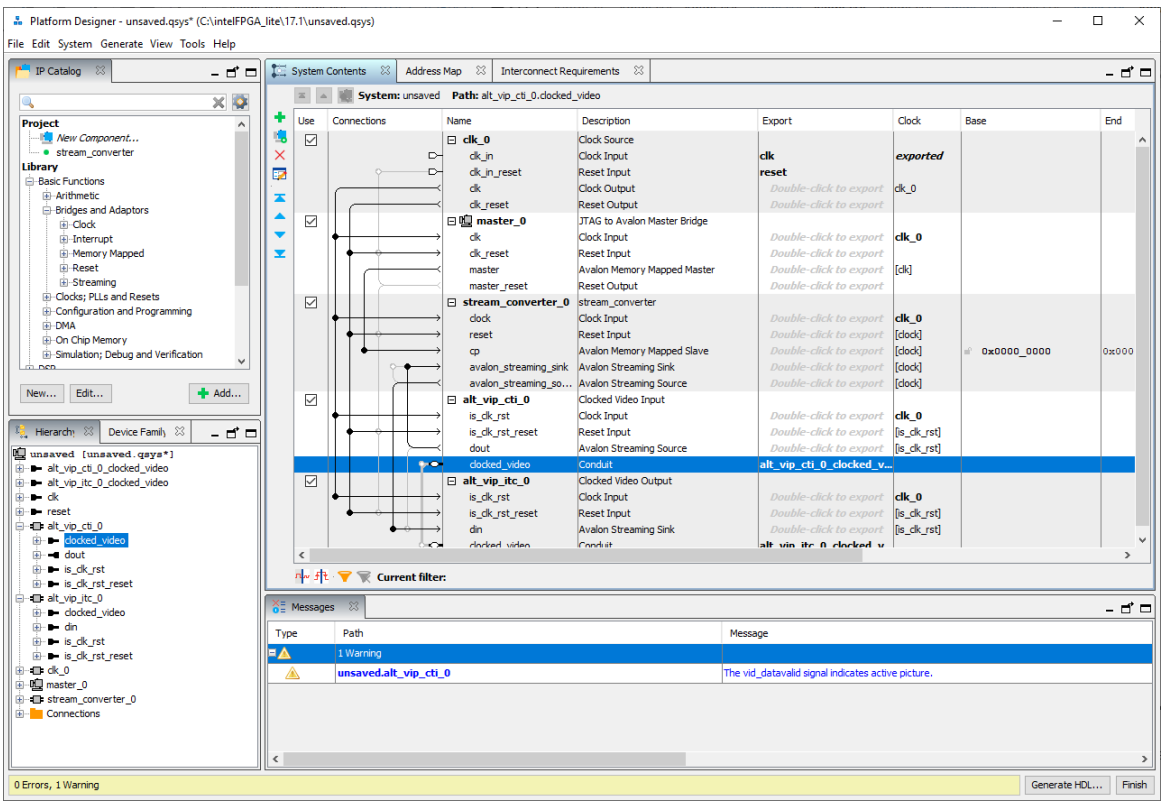

Рисунок 13 – Собранная тестовая система

Заключение

Аналогичным образом можно использовать шину AXI, а если новый модуль имеет какие-то нестандартные интерфейсы, или просто те, которые необходимо экспортировать на физические пины микросхемы, то можно выбрать в качестве типа интерфейса conduit, который не регламентирует набор сигналов в интерфейсе.

В примере выше мы создавали IP-ядро с помощью графической оболочки, и в результате у нас сформировался файл stream_converter_hw.tcl, который описывает все наши настройки в текстовом виде. Для того, чтобы в новом проекте было доступно уже созданное IP-ядро, необходимо в настройках (Tools ->Options) добавить путь к файлу stream_converter_hw.tcl.

Как вы уже поняли, мы могли бы вместо создания IP ядра в графической среде все сделать в текстовом редакторе, т. к. файл stream_converter_hw.tcl является просто скриптом на языке TCL. Но я предпочитаю создавать IP-ядра в графической среде по нескольким причинам. Во-первых, не надо помнить свойства всех интерфейсов (т. к. в реальной работе создавать IP-ядра для Platform Designer приходится нечасто, то все свойства интерфейсов быстро забываются). А во-вторых, PlatformDesigner в реальном времени указывает нам на ошибки.