Продолжение. Начало в № 2`2007.

Разработка базовых компонентов цифровых устройств, реализуемых на базе ПЛИС FPGA фирмы Xilinx®, с помощью генератора параметризированных модулей CORE Generator

Валерий ЗОТОВ
walerry@km.ru

Генерация описаний комплексных умножителей на основе параметризированного модуля Complex Multiplier с помощью средств CORE Generator

Для подготовки описаний устройств, выполняющих операцию перемножения двух комплексных значений, в составе средств CORE Generator предусмотрен параметризированный модуль *Complex Multiplier*. Значения входных данных в комплексных умножителях (операндов в комплексной форме) представляются в виде:

$$A = Ar + jAi, \quad (1)$$
$$B = Br + jBi, \quad (2)$$

где Ar и Br — вещественная часть значений входных данных A и B соответственно; Ai и Bi — мнимая часть операндов A и B соответственно.

Результат выполнения операции комплексного умножения можно записать в виде следующего выражения:

$$P = A \times B = Pr + jPi. \quad (3)$$

В этом выражении значения вещественной (Pr) и мнимой (Pi) частей произведения P определяются по формулам (4) и (5):

$$Pr = Ar \times Br - Ai \times Bi, \quad (4)$$
$$Pi = Ar \times Bi + Ai \times Br. \quad (5)$$

Данные выражения могут непосредственно применяться для вычисления значения комплексного произведения в создаваемом устройстве. Для реализации такого метода необходимо четыре аппаратных блока умножения (18×18 Multiplier Blocks или XtremeDSP). Структурное представление архитектуры формируемого комплексного умножителя, использующего этот способ вычислений, показано на рис. 73.

С целью минимизации количества используемых аппаратных блоков умножения выражения (4) и (5) можно преобразовать к следующему виду:

$$Pr = Ar \times (Br + Bi) - (Ar + Ai) \times Br, \quad (6)$$
$$Pi = Ar \times (Br + Bi) + (Ai - Ar) \times Br. \quad (7)$$

При использовании преобразованных выражений (6) и (7) в качестве алгоритма вычисления вещественной и мнимой части произведения для реализации комплексного умножителя достаточно трех аппаратных блоков умножения. В этом случае необходимы дополнительные сумматоры и вычитающие устройства, которые выполняются на основе ресурсов стандартных логических ячеек ПЛИС. Структурное представление архитектуры комплексного умножителя, реализуемого на основе трех аппаратных блоков умножения, приведено на рис. 74.

Умножители комплексных значений находят широкое применение в составе различных цифровых систем. Одной из наиболее заметных областей применения комплексных умножителей является разработка устройств цифровой обработки сигналов и, в первую очередь, проектирование цифровых фильтров. Поэтому параметризированный модуль *Complex Multiplier* включен

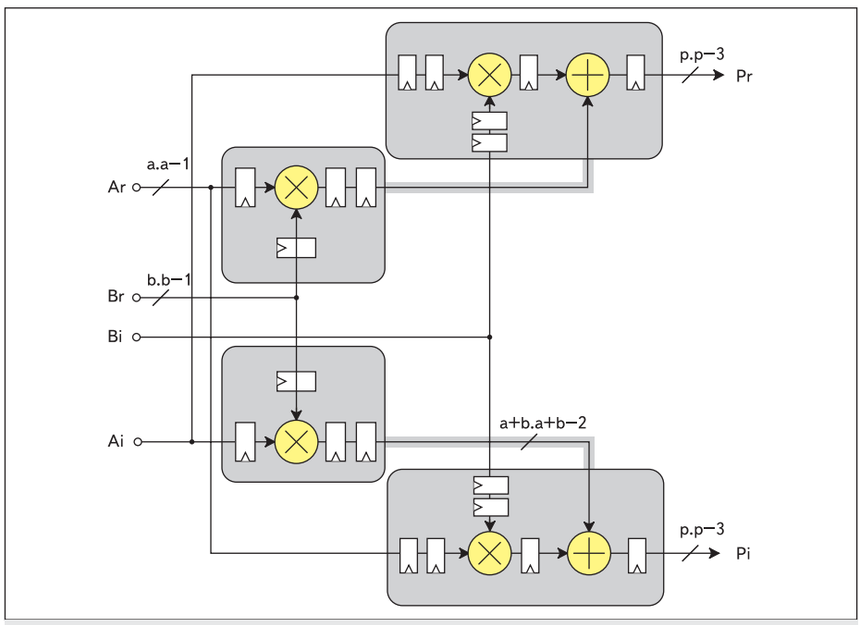

Рис. 73. Структурное представление архитектуры формируемого комплексного умножителя, выполняемого на основе четырех аппаратных блоков умножения

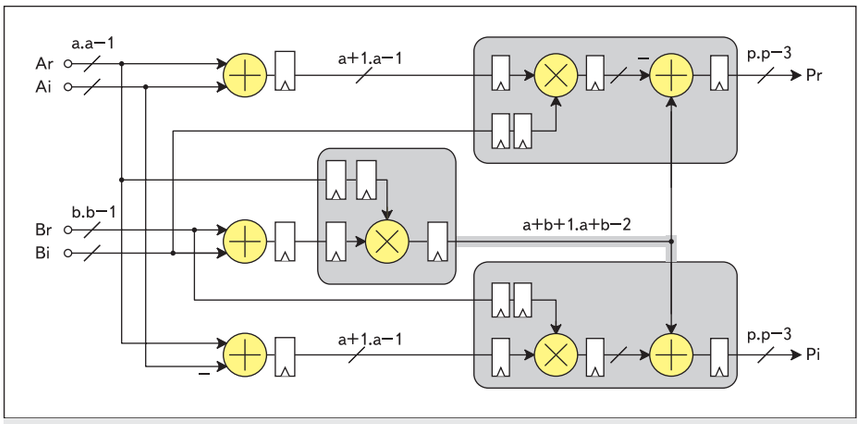

Рис. 74. Структурное представление архитектуры формируемого комплексного умножителя, выполняемого на основе трех аппаратных блоков умножения

не только в состав группы ядер *Math Functions*, предназначенных для реализации математических функций, но и в категорию ядер *Digital Signal Processing*, используемых в устройствах цифровой обработки сигналов. В рассматриваемой версии генератора параметризированных модулей CORE Generator представлено несколько модификаций данного ядра, среди которых наиболее широкими возможностями обладает вариант *Complex Multiplier v2.1*. Указанная версия параметризированного модуля позволяет создавать описания умножителей комплексных значений, которые предназначены для последующей реализации на базе кристаллов семейств Spartan-3; Virtex-II; Virtex-II PRO и Virtex-4.

Ядро комплексного умножителя *Complex Multiplier* версии v2.1. обладает следующими функциональными возможностями:

- оптимальное использование ресурсов аппаратных блоков XtremeDSP;
- поддержка различных вариантов реализации генерируемых элементов (с применением трех или четырех аппаратных блоков умножения);
- индивидуальный выбор разрядности каждого комплексного сомножителя (входных шин данных) в диапазоне от 4 до 63 двоичных разрядов;
- поддержка разрядности вычисляемого комплексного произведения в диапазоне от 3 до 127 двоичных разрядов;
- возможность выбора метода округления (усечения) вычисляемого комплексного произведения;
- поддержка двух критериев оптимизации формируемых умножителей комплексных значений;
- возможность выбора параметров конвейерной организации выполнения операций, позволяющей повысить производительность генерируемого комплексного умножителя;
- наличие синхронного входа сброса, использование которого определяется по выбору разработчика.

«Мастер» настройки параметров ядра *Complex Multiplier* версии v2.1 содержит единственную диалоговую панель, в которой основной является страница *Parameters*. Вид этой страницы показан на рис. 75.

Процесс определения параметров формируемого комплексного умножителя, как правило, начинается с ввода названия типа создаваемого элемента в поле редактирования *Component Name*. Затем нужно указать требуемые значения разрядности операндов (входных шин данных) и результата выполнения операции комплексного умножения (выходных шин). Для этого нужно воспользоваться полями редактирования, которые расположены во встроенной панели *Data Width* (рис. 75). Значения разрядности первого и второго сомножителя определяются с помощью полей редактирования *Operand A* и *Operand B* соответственно. Количество разрядов двоичного кода, представляющего значение вычисляемого комплексного произведения, указывается в поле редактирования *Product*. Информация о границах диапазона допустимых значений этих параметров отображается в строках *Valid Range*, которые располагаются справа от соответствующих полей редактирования. Следует обратить внимание на то, что верхняя граница диапазона допустимых значений количества разрядов в двоичном коде вычисляемого комплексного произведения *Product* изменяется динамически при вводе новых значений разрядности входных операндов *Operand A* и *Operand B*. Максимальное значение числа разрядов, используемых для представления результата выполнения операции комплексного умножения, равно сумме значений разрядности сомножителей, увеличенной на единицу. Это значение соответствует количеству разрядов выходных шин, которое необходимо для точного представления результата выполнения операций комплексного умножения.

Если в поле редактирования *Product* указывается значение разрядности выходных шин данных, которое меньше, чем максимально допустимое значение этого параметра, то результат выполнения операций в формируемом комплексном умножителе будет представлен в округленном или усеченном виде. В этом случае нужно выбрать один из двух возможных методов округления/усечения вычисляемого комплексного произведения, используя пару кнопок с зависимой фиксацией, которые находятся во встроенной панели *Rounding Mode* (рис. 75). Если в нажатом состоянии зафиксирована кнопка *Round Results*, то в формируемом комплексном умножителе будет использоваться стандартный способ округления. При этом в состав интерфейса создаваемого элемента автоматически добавляется дополнительный вход переноса

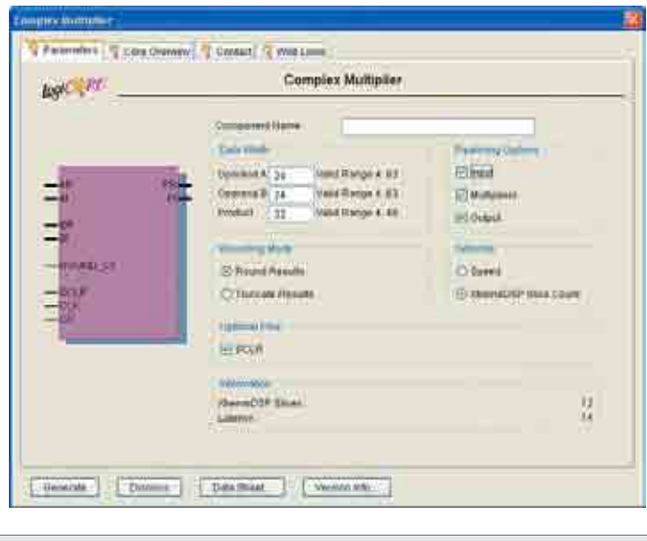

Рис. 75. Страница Parameters диалоговой панели «мастера» настройки параметров ядра комплексного умножителя Complex Multiplier версии v2.1

ROUND_CY. Значение сигнала на этом входе используется при округлении результата выполнения операции комплексного умножения в том случае, когда значение округляемой дробной части (неиспользуемых младших разрядов) равно 0,5. Если в нажатое положение переключена кнопка *Truncate Results*, то в генерируемом комплексном умножителе будет выполняться усечение полученного результата до указанного количества двоичных разрядов, при котором младшие неиспользуемые биты значения отбрасываются, а оставшиеся разряды поступают на выход в неизменном виде. При выборе метода округления/усечения вычисляемого комплексного произведения следует учитывать, что округление результата вычислений может в некоторых случаях привести к увеличению объема используемых ресурсов ПЛИС и к появлению дополнительной задержки в процессе вычислений.

Далее следует определить параметры конвейерной организации обработки данных в разрабатываемом умножителе комплексных значений. Для этой цели предназначены индикаторы состояния *Input*, *Multipliers*, *Output*, представленные во встроенной панели *Pipelining Options* (рис. 75). Каждый из этих индикаторов состояния позволяет добавить соответствующую группу регистров в состав структуры генерируемого элемента. Для применения в составе структуры формируемого комплексного умножителя входных регистров нужно перевести во включенное состояние индикатор *Input*. Если необходимо задействовать внутренние конвейерные регистры в умножителях, используемых в составе формируемого элемента, то следует установить индикатор *Multipliers* в состояние «Включено». Чтобы включить в состав структуры создаваемого комплексного умножителя выходные регистры, нужно перевести во включенное состояние индикатор *Output*.

Выбор дополнительного входа управления в генерируемом умножителе комплексных значений осуществляется с помощью индикатора состояния, который расположен во встроенной панели *Optional Pin*. Чтобы задействовать вход синхронного сброса, нужно установить в состояние «Включено» индикатор *SCLR*. При наличии активного уровня сигнала (высокого логического уровня) на этом входе содержимое внутренних регистров и триггеров комплексного умножителя обнуляется по фронту тактового сигнала. Выходы умножителя также устанавливаются в нулевое состояние.

Завершая процесс определения параметров формируемого умножителя комплексных значений, следует выбрать один из двух критериев дополнительной оптимизации. Выбор требуемого критерия осуществляется с помощью двух кнопок с зависимой фиксацией, которые находятся во встроенной панели *Optimize* (рис. 75). Чтобы установить в качестве критерия оптимизации генерируемого комплексного умножителя достижение максимальной производительности, нужно переключить в нажатое состояние кнопку *Speed*. Для минимизации количества аппаратных блоков XtremeDSP или 18×18 Multiplier Blocks, требуемых для реализации формируемого умножителя комплексных значений, следует зафиксировать в нажатом положении кнопку *XtremeDSP Slice Count* или *18×18 Multiplier* в зависимости от используемого семейства ПЛИС.

Сведения о числе аппаратных блоков XtremeDSP или 18×18 Multiplier Blocks, используемых для реализации генерируемого умножителя, приводятся во встроенной информационной панели *Information*, которая расположена в нижней части диалоговой панели «мастера» настройки параметров ядра *Complex Multiplier* версии v2.1. В этой же встроенной информационной панели отображается значение задержки вычисления комплексного произведения *Latency*, выраженное в количестве тактовых периодов. Значение этой задержки зависит от количества ступеней конвейерной организации вычислений (конвейерных регистров) в создаваемом элементе.

В описаниях комплексных умножителей, формируемых с помощью параметризированного модуля *Complex Multiplier* версии v2.1, применяется следующая система условных обозначений входных и выходных портов:

- ar [K:0] — входная шина данных с разрядностью K+1, определяющих значение вещественной части первого сомножителя Ar;
- ai [K:0] — входная шина данных с разрядностью K+1, определяющих значение мнимой части первого сомножителя Ai;
- br [M:0] — входная шина данных с разрядностью M+1, определяющих значение вещественной части второго сомножителя Br;
- bi [M:0] — входная шина данных с разрядностью M+1, определяющих значение мнимой части второго сомножителя Bi;
- pr[N:0] — выходная шина данных с разрядностью N+1, определяющих значение вещественной части полученного произведения Pr;
- pi[N:0] — выходная шина данных с разрядностью N+1, определяющих значение мнимой части полученного произведения Pi;
- round_cy — вход сигнала переноса, используемого при округлении полученного результата операции комплексного умножения;
- clk — вход тактового сигнала;
- ce — вход сигнала разрешения синхронизации;
- sclr — вход сигнала синхронного сброса.

Из приведенной выше совокупности идентификаторов в описании интерфейса сформированного умножителя комплексных значений будут присутствовать условные обозначения только тех входных портов, которые были указаны разработчиком при определении параметров этого элемента.

Пример описания комплексного умножителя, сформированного на основе параметризированного модуля Complex Multiplier версии v2.1

В качестве примера описания комплексного умножителя, сформированного на основе параметризированного модуля *Complex Multiplier* версии v2.1 с помощью средств CORE Generator, далее приведен текст VHDL-описания элемента cmplx_multiplier_32. Этот элемент предназначен для выполнения операции умножения двух комплексных 32-разрядных значений, представленных на его входных шинах. Вычисляемое значение произведения отображается на выходах комплексного умножителя в форме 48-разрядного двоичного кода, полученного в результате округления точного значения. В сформированном комплексном умножителе cmplx_multiplier_32 используется максимальное количество конвейерных регистров, а также задействован вход синхронного сброса:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY cmplx_multiplier_32 IS
   port (
         ar: IN std_logic_VECTOR(31 downto 0);
         ai: IN std_logic_VECTOR(31 downto 0);
         br: IN std_logic_VECTOR(31 downto 0);
         bi: IN std_logic_VECTOR(31 downto 0);
         round_cy: IN std_logic;
         pr: OUT std_logic_VECTOR(47 downto 0);
         pi: OUT std_logic_VECTOR(47 downto 0);
         clk: IN std_logic;
         ce: IN std_logic;
         sclr: IN std_logic
         );
END cmplx_multiplier_32;
--
ARCHITECTURE cmplx_multiplier_32_a OF cmplx_multiplier_32
IS
-- synopsys translate_off
component wrapped_cmplx_multiplier_32
   port (
         ar: IN std_logic_VECTOR(31 downto 0);
         ai: IN std_logic_VECTOR(31 downto 0);
         br: IN std_logic_VECTOR(31 downto 0);
         bi: IN std_logic_VECTOR(31 downto 0);
         round_cy: IN std_logic;
         pr: OUT std_logic_VECTOR(47 downto 0);
         pi: OUT std_logic_VECTOR(47 downto 0);
         clk: IN std_logic;
         ce: IN std_logic;
         sclr: IN std_logic
         );
end component;
--
-- Configuration specification
   for all : wrapped_cmplx_multiplier_32 use entity
XilinxCoreLib.cmpy_v2_1(behavioral)
         generic map(
                        pipe_in => 1,
                        c_family => «virtex4»,
                        b_width => 32,
                        pipe_mid => 1,
                        m_width => -1,
                        c_enable_rlocs => 1,
                        optimize => 0,
                        c_has_sclr => 1,
                        a_width => 32,
                        pipe_out => 1,
                        round => 1,
                        p_width => 48
                        );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_cmplx_multiplier_32
```

```
        port map (
                ar => ar,
                ai => ai,
                br => br,
                bi => bi,
                round_cy => round_cy,
                pr => pr,
                pi => pi,
                clk => clk,
                ce => ce,
                sclr => sclr
                );
-- synopsys translate_on
--
END cmplx_multiplier_32_a;
```

Для декларации созданного компонента комплексного умножителя cmplx_multiplier_32 необходимо включить в состав соответствующего раздела VHDL-описания разрабатываемого устройства следующую последовательность выражений:

```
component cmplx_multiplier_32
   port (
        ar: IN std_logic_VECTOR(31 downto 0);
        ai: IN std_logic_VECTOR(31 downto 0);
        br: IN std_logic_VECTOR(31 downto 0);
        bi: IN std_logic_VECTOR(31 downto 0);
        round_cy: IN std_logic;
        pr: OUT std_logic_VECTOR(47 downto 0);
        pi: OUT std_logic_VECTOR(47 downto 0);
        clk: IN std_logic;
        ce: IN std_logic;
        sclr: IN std_logic
        );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of cmplx_multiplier_32: component is
«true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of cmplx_multiplier_32: component is true;
```

Создание экземпляра умножителя комплексных значений с архитектурой компонента cmplx_multiplier_32 в составе описания проектируемого устройства осуществляется с помощью оператора, шаблон которого выглядит следующим образом:

```
<идентификатор_экземпляра_комплексного_умножителя> : cm-
plx_multiplier_32
          port map (
                ar => ar,
                ai => ai,
                br => br,
                bi => bi,
                round_cy => round_cy,
                pr => pr,
                pi => pi,
                clk => clk,
                ce => ce,
                sclr => sclr
                );
```

Формирование описаний элементов, выполняющих операцию деления, на основе параметризированного модуля Pipelined Divider версии v3.0 с помощью средств CORE Generator

Рассматриваемая версия генератора параметризированных модулей CORE Generator поддерживает два ядра *Pipelined Divider* версии v3.0 и *Divider Generator* версии v1.0, которые предназначены для формирования описаний элементов, выполняющих операцию деления. Указанные параметризированные модули включены в состав группы ядер *Math Functions*, предназначенных для реализации математических функций. Каждый из этих параметризированных модулей предоставляет разработчику различные возможности, поэтому далее поочередно рассматриваются особенности указанных ядер.

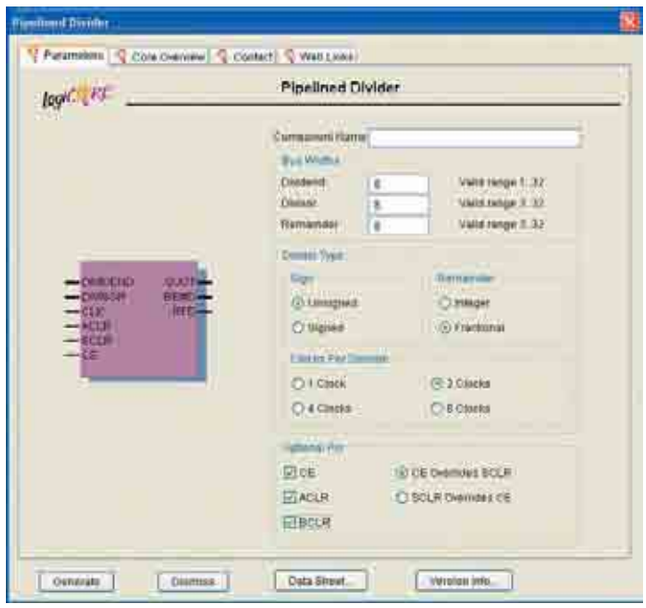

Рис. 76. Страница Parameters диалоговой панели «мастера» настройки параметров ядра делителя Pipelined Divider версии v3.0

- возможность реализации операций целочисленного и дробного деления;
- поддержка операций деления, выполняемых над входными данными, представленными в виде значений со знаком или без знака;
- применение конвейерного метода выполнения операций, позволяющего повысить производительность генерируемого устройства;
- возможность индивидуального выбора разрядности делимого и делителя (входных шин данных) в диапазоне от 1 до 32 и от 3 до 32 двоичных разрядов соответственно;
- поддержка разрядности вычисляемого дробного остатка в диапазоне от 3 до 32 двоичных разрядов;
- возможность реализации обратной функции (1/X);
- реализация формируемых элементов в виде синхронных устройств, тактируемых одним сигналом синхронизации;
- использование регистров на выходах формируемого элемента;
- возможность выборочного использования в создаваемых элементах синхронного и асинхронного входов сброса, а также входа сигнала разрешения синхронизации.

Выбор типа устройства деления, формируемого на основе параметризированного модуля *Pipelined Divider* версии v3.0, и определение его параметров осуществляется с помощью «мастера» настройки этого ядра, который содержит одну диалоговую панель. На рис. 76 показан вид основной страницы *Parameters* данной диалоговой панели.

В представленной диалоговой панели после ввода названия типа создаваемого элемента (в поле редактирования *Component Name*) нужно указать значения разрядности используемых операндов (входных и выходных шин данных). Для этой цели предназначены поля редактирования, расположенные во встроенной панели *Bus Widths* (рис. 76). Количество разрядов первой входной шины данных, совокупность уровней сигналов которой образует значение делимого, указывается в поле редактирования *Dividend*. Число разрядов второй входной шины данных, состояние сигналов которой определяет значение делителя, задается в поле редактирования *Divisor*. Разрядность первой выходной шины данных, предназначенной для отображения значения целой части результата

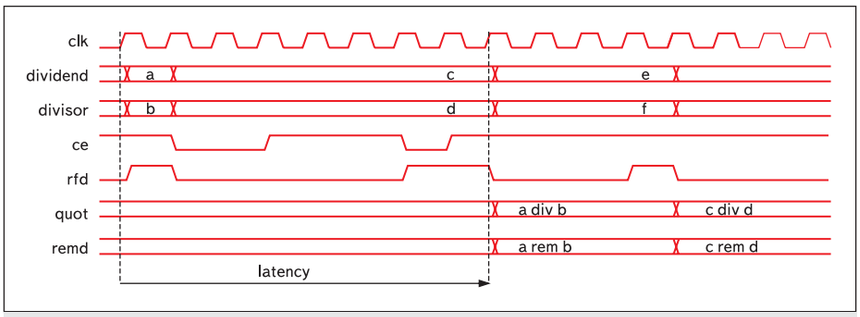

Рис. 77. Временные диаграммы, поясняющие выполнение операции деления в элементах, генерируемых на основе параметризированного модуля Pipelined Divider версии v3.0

операции деления, автоматически устанавливается равной разрядности входной шины данных, определяющих значение делимого. Количество разрядов второй выходной шины данных, представляющей значение остатка или дробной части результата операции деления в двоичном коде, указывается в поле редактирования *Remainder*.

Далее, используя три группы кнопок с зависимой фиксацией, которые находятся во встроенной панели *Divider Type* (рис. 76), нужно выбрать тип формируемого элемента, предназначенного для выполнения операции деления. Прежде всего, рекомендуется определить форму представления значений входных и выходных данных, воспользовавшись группой кнопок с зависимой фиксацией *Sign*. Если необходимо подготовить описание элемента, выполняющего деление значений входных данных без знака, то в нажатом состоянии должна быть зафиксирована кнопка *Unsigned*. Чтобы сформировать описание делителя, входные данные которого интерпретируются как значения со знаком, нужно переключить в нажатое положение кнопку *Signed*.

Затем с помощью группы кнопок с зависимой фиксацией *Remainder* следует выбрать вид операции деления, которую должен выполнять создаваемый элемент. Для генерации описания элемента, осуществляющего операцию целочисленного деления (деления с остатком) необходимо зафиксировать в нажатом состоянии кнопку *Integer*. Чтобы сформировать описание элемента, выполняющего операцию дробного деления, нужно переключить в нажатое положение кнопку *Fractional*.

Группа кнопок с зависимой фиксацией *Clocks Per Division* позволяет указать количество периодов тактового сигнала, которое отводится для выполнения операции деления в создаваемом элементе. При этом учитываются только те периоды тактового сигнала, в течение которых на входе сигнала разрешения синхронизации Clock Enable (CE) присутствует напряжение высокого логического уровня. Значение параметра *Clocks Per Division* оказывает влияние на величину суммарной задержки появления нового результата вычислений на выходах этого элемента после изменения входных данных. Увеличение числа периодов тактового сигнала, выделяемого для выполнения операции деления, приводит к возрастанию длительности общей задержки формирования выходных значений. Но при этом сокращается объем ресурсов ПЛИС, требуемых для реализации создаваемого элемента. Разработчик может выбрать один из четырех возможных значений параметра *Clocks Per Division* — 1, 2, 4 или 8 периодов тактового сигнала, переключив в нажатое состояние кнопку *1 Clocks*, *2 Clocks*, *4 Clocks* или *8 Clocks* соответственно. На рис. 77 приведены временные диаграммы, поясняющие выполнение операции деления в элементах, генерируемых на основе параметризированного модуля *Pipelined Divider* версии v3.0, при значении параметра *Clocks Per Division* равном *4 Clocks*.

Завершает процесс установки значений параметров создаваемого элемента, осуществляющего операцию деления, определение требуемых дополнительных входов управляющих сигналов. Для этой цели нужно воспользоваться индикаторами состояния, которые представлены во встроенной панели *Optional Pin* (рис. 76). Для использования в генерируемом элементе входа асинхронного сброса необходимо установить во включенное состояние индикатор *ACLR*. Чтобы задействовать вход синхронного сброса, следует перевести в состояние «Включено» индикатор *SCLR*. Если в создаваемом элементе необходим вход сигнала разрешения синхронизации Clock Enable, то нужно воспользоваться индикатором состояния *CE*, находящимся в этой же встроенной панели. При одновременном использовании входов синхронного сброса и разрешения синхронизации необходимо определить приоритеты соответствующих сигналов на этих входах с помощью двух кнопок с зависимой фиксацией. Чтобы сигнал на входе разрешения синхронизации имел более высокий приоритет по сравнению с сигналом на входе синхронного сброса, следует зафиксировать в нажатом состоянии кнопку *CE Overrides SCLR*. Для установки противоположного соотношения приоритетов указанных сигналов нужно переключить в нажатое состояние кнопку *SCLR Overrides CE*.

В VHDL-описаниях элементов, предназначенных для выполнения операции деления и подготовленных с помощью параметризированного модуля *Pipelined Divider* версии v3.0, применяется следующая система условных обозначений интерфейсных портов:

- dividend [M:0] — входная шина данных с разрядностью M+1, совокупность сигналов которой определяет значение делимого;
- divisor [K:0] — входная шина данных с разрядностью K+1, совокупность сигналов которой определяет значение делителя;
- quot [M:0] — выходная шина данных с разрядностью M+1, совокупность сигналов которой определяет значение целой части результата операции деления;
- remd [N:0] — выходная шина данных с разрядностью N+1, совокупность сигналов которой определяет значение остатка или дробной части результата операции деления;
- clk — вход тактового сигнала;
- rfd — выход сигнала Ready for Data (RFD), информирующего о завершении выборки новых входных данных;
- aclr — вход сигнала асинхронного сброса;
- sclr — вход сигнала синхронного сброса;
- ce — вход сигнала разрешения синхронизации.

Пример описания элемента, предназначенного для выполнения операции дробного деления, сформированного на основе параметризированного модуля Pipelined Divider версии v3.0

Примером описания устройства, используемого для выполнения операции дробного деления, сформированного на основе параметризированного модуля *Pipelined Divider* версии v3.0 с помощью средств CORE Generator, является VHDL-описание элемента divider_18_16, текст которого приведен далее. Этот элемент предназначен для вычисления результата деления 18-разрядного значения делимого на 16-разрядное значение делителя, которые представлены в виде соответствующего двоичного кода на входных шинах данных. Полученный результат выполнения операции (частное) отображается на выходных шинах сгенерированного элемента в двоичном коде в виде 18-разрядного значения целой части и 16-разрядного значения дробной части. В сформированном элементе divider_18_16 задействованы входы синхронного и асинхронного сброса и вход разрешения синхронизации:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY divider_18_16 IS
  port (
        dividend: IN std_logic_VECTOR(17 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quot: OUT std_logic_VECTOR(17 downto 0);
```

```
        remd: OUT std_logic_VECTOR(15 downto 0);
        clk: IN std_logic;
        rfd: OUT std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic
        );
END divider_18_16;
--
ARCHITECTURE divider_18_16_a OF divider_18_16 IS
-- synopsys translate_off
component wrapped_divider_18_16
   port (
        dividend: IN std_logic_VECTOR(17 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quot: OUT std_logic_VECTOR(17 downto 0);
        remd: OUT std_logic_VECTOR(15 downto 0);
        clk: IN std_logic;
        rfd: OUT std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic
        );
end component;
--
-- Configuration specification
   for all : wrapped_divider_18_16 use entity XilinxCoreLib.sdi-
vider_v3_0(behavioral)
        generic map(
                c_has_ce => 1,
                divclk_sel => 4,
                dividend_width => 18,
                fractional_width => 16,
                c_sync_enable => 1,
                signed_b => 1,
                divisor_width => 16,
                c_has_aclr => 1,
                fractional_b => 1,
                c_has_sclr => 1
                );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_divider_18_16
        port map (
                dividend => dividend,
                divisor => divisor,
                quot => quot,
                remd => remd,
                clk => clk,
                rfd => rfd,
                aclr => aclr,
                sclr => sclr,
                ce => ce
                );
-- synopsys translate_on
--
END divider_18_16_a;
```

При использовании представленного выше элемента divider_18_16 в качестве компонента проектируемого устройства нужно включить в состав формируемого VHDL-описания этого устройства следующие выражения декларации:

```
component divider_18_16
   port (
        dividend: IN std_logic_VECTOR(17 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quot: OUT std_logic_VECTOR(17 downto 0);
        remd: OUT std_logic_VECTOR(15 downto 0);
        clk: IN std_logic;
        rfd: OUT std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic
        );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of divider_18_16: component is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of divider_18_16: component is true;
```

Для создания экземпляра компонента divider_18_16 в описании разрабатываемого устройства нужно добавить в состав архитектурного блока оператор, шаблон которого выглядит следующим образом:

```
<идентификатор_делителя_divider_18_16> : divider_18_16
        port map (
                dividend => dividend,
                divisor => divisor,
                quot => quot,
                remd => remd,
                clk => clk,
                rfd => rfd,
                aclr => aclr,
                sclr => sclr,
                ce => ce
                );
```

Пример описания элемента, предназначенного для выполнения операции целочисленного деления, сформированного на основе параметризированного модуля Pipelined Divider версии v3.0

В качестве примера, который демонстрирует результат применения параметризированного модуля *Pipelined Divider* версии v3.0 для генерации описания устройства, выполняющего операцию целочисленного деления (деления с остатком), далее приводится текст VHDL-описания элемента divider_24_18, сформированного с помощью средств CORE Generator. Данный элемент предназначен для вычисления целого результата (неполного частного) и остатка от деления 20-разрядного значения делимого на 18-разрядное значение делителя. Значение результата выполнения операции целочисленного деления отображается в виде 24-разрядного двоичного кода на выходной шине quot сформированного элемента. Значение остатка представлено в виде 18-разрядного двоичного кода на выходной шине remd:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY divider_24_18 IS
   port (
        dividend: IN std_logic_VECTOR(23 downto 0);
        divisor: IN std_logic_VECTOR(17 downto 0);
        quot: OUT std_logic_VECTOR(23 downto 0);
        remd: OUT std_logic_VECTOR(17 downto 0);
        clk: IN std_logic;
        rfd: OUT std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic
        );
END divider_24_18;
--
ARCHITECTURE divider_24_18_a OF divider_24_18 IS
-- synopsys translate_off
component wrapped_divider_24_18
   port (
        dividend: IN std_logic_VECTOR(23 downto 0);
        divisor: IN std_logic_VECTOR(17 downto 0);
        quot: OUT std_logic_VECTOR(23 downto 0);
        remd: OUT std_logic_VECTOR(17 downto 0);
        clk: IN std_logic;
        rfd: OUT std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic
        );
end component;
--
-- Configuration specification
   for all : wrapped_divider_24_18 use entity XilinxCoreLib.sdi-
vider_v3_0(behavioral)
        generic map(
                c_has_ce => 1,
                divclk_sel => 1,
                dividend_width => 24,
                fractional_width => 18,
                c_sync_enable => 1,
                signed_b => 0,
                divisor_width => 18,
                c_has_aclr => 1,
                fractional_b => 0,
                c_has_sclr => 1
                );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_divider_24_18
        port map (
                dividend => dividend,
                divisor => divisor,
                quot => quot,
                remd => remd,
                clk => clk,
                rfd => rfd,
                aclr => aclr,
                sclr => sclr,
                ce => ce
                );
-- synopsys translate_on
--
END divider_24_18_a;
```

Для применения элемента divider_24_18 в качестве компонента разрабатываемого устройства следует включить в состав VHDL-описания этого устройства приведенную далее последовательность выражений:

```
component divider_24_18
   port (
        dividend: IN std_logic_VECTOR(23 downto 0);
        divisor: IN std_logic_VECTOR(17 downto 0);
        quot: OUT std_logic_VECTOR(23 downto 0);
        remd: OUT std_logic_VECTOR(17 downto 0);
        clk: IN std_logic;
        rfd: OUT std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic
        );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of divider_24_18: component is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of divider_24_18: component is true;
```

Чтобы добавить экземпляр компонента divider_24_18 в описание проектируемого устройства, нужно включить в состав этого описания оператор, шаблон которого имеет следующий вид:

```
<идентификатор_делителя_ divider_24_18> : divider_24_18
        port map (
                dividend => dividend,
                divisor => divisor,
                quot => quot,
                remd => remd,
                clk => clk,
                rfd => rfd,
                aclr => aclr,
                sclr => sclr,
                ce => ce
                );
```

Подготовка описаний элементов, выполняющих операцию деления, на основе параметризированного модуля Divider Generator версии v1.0 с помощью средств CORE Generator

Параметризированный модуль *Divider Generator* версии v1.0 предназначен для генерации описаний элементов, выполняющих

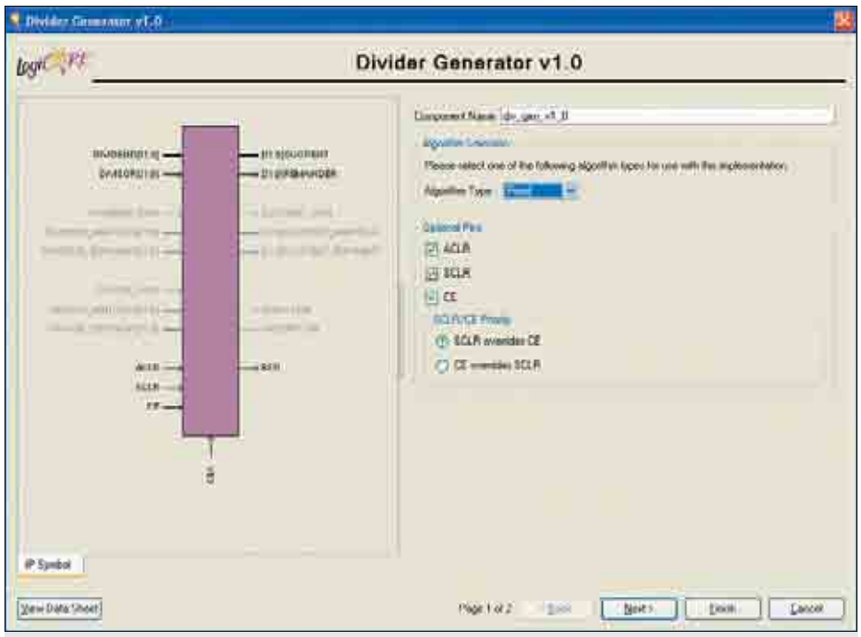

Рис. 78. Стартовая диалоговая панель «мастера» настройки параметров ядра Divider Generator версии v1.0

операцию деления входных данных, которые можно применять в составе проектов, реализуемых на основе кристаллов следующих семейств: Spartan-II; Spartan-IIE; Spartan-3; Spartan-3E; Virtex; QPRO Virtex Rad-Hard; QPRO Virtex Hi-Rel; Virtex-E; QPRO Virtex-E Military; Virtex-II; Virtex-II PRO и Virtex-4. Главным отличием этого ядра от параметризированного модуля *Pipelined Divider* версии *v3.0*, рассмотренного в предыдущих разделах, является возможность создания описаний элементов, осуществляющих операцию деления значений, представленных в формате как с фиксированной, так и с плавающей запятой. При использовании формата с плавающей запятой поддерживается представление входных данных и результата выполнения операции в соответствии со стандартом IEEE Std 754.

Определение типа и необходимых параметров элемента, формируемого на основе параметризированного модуля *Divider Generator* версии v1.0, производится с помощью соответствующего «мастера» настройки, который содержит две диалоговые панели. Стартовая диалоговая панель этого «мастера», вид которой показан на рис. 78, предназначена для выбора вида выполняемой операции деления (формы представления операндов и полученного результата): с фиксированной или плавающей запятой и дополнительных входов управления.

Вид выполняемой операции деления указывается с помощью поля выбора *Algorithm Type*, которое расположено во встроенной панели *Algorithm Selection*. Чтобы создаваемый элемент выполнял операцию деления входных значений, представленных в формате с плавающей запятой, в выпадающем списке этого поля выбора нужно выделить вариант *Float*. Для формирования описания элемента, осуществляющего операцию деления над входными данными, представленными в формате с фиксированной запятой, в выпадающем списке возможных вариантов *Algorithm Type* следует выбрать строку *Fixed*.

Для выбора дополнительных входов управления в создаваемом элементе следует воспользоваться индикаторами состояния *ACLR*, *SCLR* и *CE*, которые находятся во встроенной панели *Optional Pins* (рис. 78). Эти индикаторы состояния позволяют добавить в состав элемента, генерируемого с помощью параметризированного модуля *Divider Generator* версии v1.0, входы асинхронного и синхронного сброса, а также вход сигнала разрешения синхронизации *Clock Enable*. В случае совместного применения входов синхронного сброса и разрешения синхронизации следует указать соотношение приоритетов соответствующих сигналов с помощью группы кнопок с зависимой фиксацией *SCLR/CE Priority*. Для назначения более высокого приоритета сигналу на входе синхронного сброса по сравнению с сигналом на входе разрешения синхронизации необходимо переключить в нажатое состояние кнопку *SCLR Overrides CE*. При установке в нажатое положение кнопки *CE Overrides SCLR* более высокий приоритет будет иметь сигнал на входе разрешения синхронизации.

Структура второй диалоговой панели «мастера» настройки параметров ядра *Divider Generator* версии v1.0 зависит от вида выполняемой операции деления, выбранного в стартовой диалоговой панели. Если создается описание элемента, предназначенного для выполнения операции деления с плавающей запятой (для параметра *Algorithm Type* выбрано значение *Float*), то вторая диалоговая панель будет иметь вид, показанный на рис. 79.

В этой диалоговой панели нужно указать требуемые значения разрядности мантиссы и показателя степени (соответствующих входных и выходных шин данных), которые используются в представлении входных данных и результата выполнения операции, а также дополнительные параметры создаваемого элемента. Количество разрядов двоичного представления мантиссы определяется в поле ре-

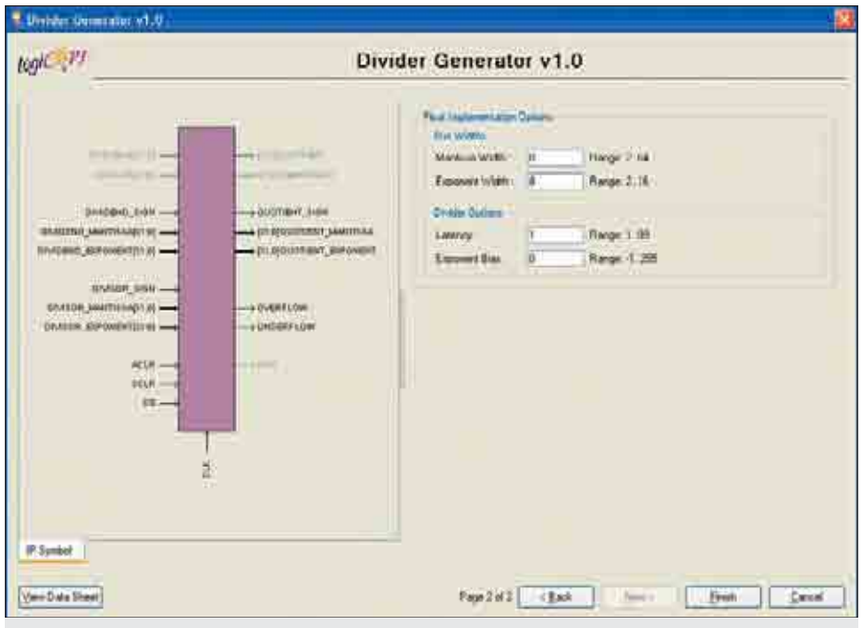

Рис. 79. Вид второй диалоговой панели «мастера» настройки параметров ядра Divider Generator версии v1.0 при создании описаний элементов, предназначенных для выполнения операции деления с плавающей запятой

дактирования *Mantissa Width*. Число разрядов двоичного кода, определяющего значение показателя степени, указывается в поле редактирования *Exponent Width*. Информация о границах допустимых диапазонов значений этих параметров приведена в строке *Range*, которая располагается справа от соответствующего поля редактирования.

Дополнительные параметры формируемого элемента определяются с помощью полей редактирования, расположенных во встроенной панели *Divider Options* (рис. 79). В поле редактирования *Latency* указывается значение задержки формирования результата операции деления, выраженное в количестве периодов тактового сигнала. При необходимости можно задать смещение показателя степени, используя поле редактирования *Exponent Bias*.

При генерации описания элемента, предназначенного для выполнения операции деления с фиксированной запятой, (для параметра *Algorithm Type* указано значение *Fixed*) вторая диалоговая панель приобретает вид, изображенный на рис. 80.

В данной панели, прежде всего, нужно определить разрядность входных шин данных, с которых считываются текущие значения операндов, с помощью полей редактирования, представленных во встроенной панели *Bus Widths* (рис. 80). Количество разрядов двоичного кода, определяющего значение делимого (входной шины данных dividend), указывается в поле редактирования *Dividend Width*. Число разрядов двоичного кода, представляющего значение делителя (входной шины данных divisor), задается в поле редактирования *Divisor Width*.

Затем с помощью полей выбора и кнопок с зависимой фиксацией, расположенных во встроенной панели *Divider Type* (рис. 80), нужно определить тип операндов и вид операции деления, выполняемой формируемым элементом. В поле выбора *Clocks Per Division* следует указать количество периодов тактового сигнала, которое отводится для выполнения операции деления в создаваемом элементе. В выпадающем списке значений этого параметра представлены те же варианты, что и для аналогичного параметра ядра *Pipelined Divider* версии v3.0, рассмотренного ранее. Тип используемых операндов (со знаком или без знака) выбирается с помощью группы кнопок с зависимой фиксацией, которые находятся во встроенной панели *Operand Sign*. Для генерации элемента, выполняющего операцию деления над входными значениями без знака, следует зафиксировать в нажатом состоянии кнопку *Unsigned*. При формировании описания делителя, в котором операнды интерпретируются как значения со знаком, необходимо переключить в нажатое положение кнопку *Signed*.

Определение вида операции деления, которую должен выполнять создаваемый элемент, осуществляется с помощью поля выбора

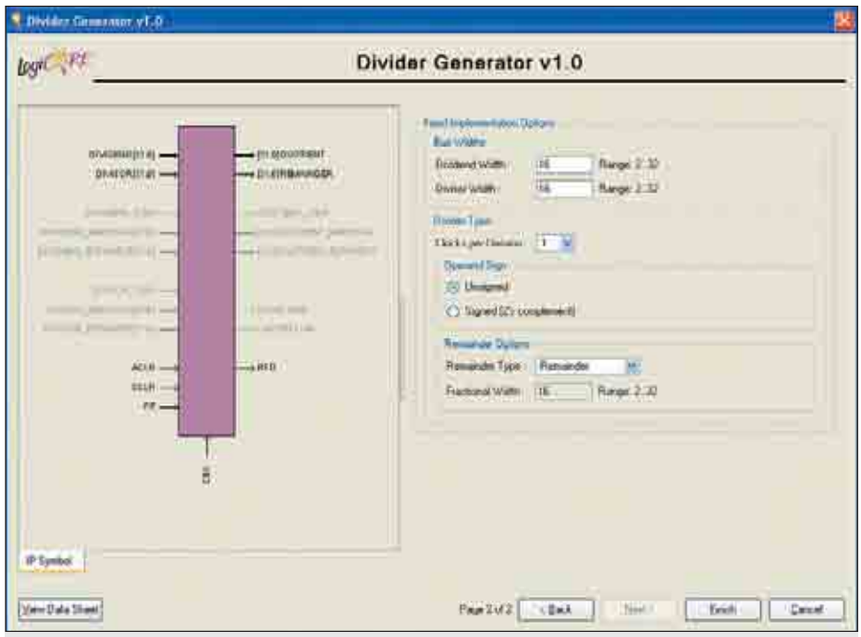

Рис. 80. Вид второй диалоговой панели «мастера» настройки параметров ядра Divider Generator версии v1.0 при создании описаний элементов, предназначенных для выполнения операции деления с фиксированной запятой

Remainder Type, которое находится во встроенной панели *Remainder Options* (рис. 80). Чтобы сформировать описание элемента, выполняющего операцию деления с остатком, следует в выпадающем списке этого поля выбрать вариант *Remainder*. Для генерации описания элемента, осуществляющего операцию дробного деления, нужно в этом списке выделить строку *Fractional*. В последнем случае станет доступным поле редактирования *Fractional Width*, расположенное в этой же встроенной панели. В данном поле необходимо указать количество разрядов двоичного кода, используемого для представления дробной части результата при выполнении операции деления с фиксированной запятой.

Система условных обозначений входных и выходных портов в описаниях элементов, формируемых на основе параметризированного модуля *Divider Generator* версии v1.0, включает в себя следующие идентификаторы:

- dividend_mantissa[K:0] — входная шина данных с разрядностью K+1, совокупность сигналов которой определяет значение мантиссы делимого при выполнении операции деления с плавающей запятой;
- dividend_sign — вход сигнала, определяющего знак делимого, представленного в формате с плавающей запятой;
- dividend_exponent[N:0] — входная шина данных с разрядностью N+1, совокупность сигналов которой определяет значение показателя степени делимого при осуществлении операции деления с плавающей запятой;
- divisor_mantissa[K:0] — входная шина данных с разрядностью K+1, совокупность сигналов которой определяет значение мантиссы делителя при выполнении операции деления с плавающей запятой;
- divisor_sign — вход сигнала, определяющего знак делителя, представленного в формате с плавающей запятой;
- divisor_exponent[N:0] — входная шина данных с разрядностью N+1, совокупность сигналов которой определяет значение показателя степени делителя при осуществлении операции деления с плавающей запятой;
- quotient_mantissa[K:0] — выходная шина данных с разрядностью K+1, совокупность сигналов которой определяет значение мантиссы полученного результата при выполнении операции деления с плавающей запятой;
- quotient_sign — выход сигнала, определяющего знак результата при осуществлении операции деления с плавающей запятой;
- quotient_exponent[N:0] — выходная шина данных с разрядностью N+1, совокупность сигналов которой определяет значение показателя степени полученного результата при выполнении операции деления с плавающей запятой;
- overflow — выход сигнала переполнения, используемый при осуществлении операции деления с плавающей запятой;
- underflow — выход сигнала исчезновения значащих разрядов, используемый при осуществлении операции деления с плавающей запятой;
- clk — вход тактового сигнала;
- ce — вход сигнала разрешения синхронизации;
- aclr — вход сигнала асинхронного сброса;

- sclr — вход сигнала синхронного сброса;
- rfd — выход сигнала Ready for Data (RFD), информирующего о завершении выборки новых входных данных;
- dividend [M:0] — входная шина данных с разрядностью M+1, совокупность сигналов которой определяет значение делимого при выполнении операции деления с фиксированной запятой;
- divisor [K:0] — входная шина данных с разрядностью K+1, совокупность сигналов которой определяет значение делителя при выполнении операции деления с фиксированной запятой;
- quotient [M:0] — выходная шина данных с разрядностью M+1, совокупность сигналов которой определяет значение целой части результата при выполнении операции деления с фиксированной запятой;
- remainder [N:0] — выходная шина данных с разрядностью N+1, совокупность сигналов которой определяет значение остатка или дробной части результата при выполнении операции деления с фиксированной запятой.

В составе описания элемента, сгенерированного с помощью параметризированного модуля *Divider Generator* версии v1.0, будут присутствовать идентификаторы только тех входных и выходных портов, которые соответствуют выбранному типу операции деления. В последующих разделах рассматриваются примеры описаний элементов для каждого поддерживаемого типа операции деления.

Пример описания элемента, предназначенного для выполнения операции деления значений с плавающей запятой, сформированного на основе параметризированного модуля Divider Generator версии v1.0

В качестве примера описания устройства, сформированного на основе параметризированного модуля *Divider Generator* версии v1.0 с помощью средств CORE Generator для выполнения операции деления входных значений с плавающей запятой, далее приводится VHDL-описание элемента div_gen_32_16_float. В этом элементе значения операндов и результата операции деления представлены в виде 32-разрядной мантиссы и 16-разрядного показателя степени. В сгенерированном элементе задействованы входы синхронного и асинхронного сброса, а также вход разрешения синхронизации:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY div_gen_32_16_float IS
   port (
         clk: IN std_logic;
         ce: IN std_logic;
         aclr: IN std_logic;
         sclr: IN std_logic;
         dividend_mantissa: IN std_logic_VECTOR(31 downto 0);
         dividend_sign: IN std_logic;
         dividend_exponent: IN std_logic_VECTOR(15 downto 0);
         divisor_mantissa: IN std_logic_VECTOR(31 downto 0);
         divisor_sign: IN std_logic;
         divisor_exponent: IN std_logic_VECTOR(15 downto 0);
         quotient_mantissa: OUT std_logic_VECTOR(31 downto 0);
         quotient_sign: OUT std_logic;
         quotient_exponent: OUT std_logic_VECTOR(15 downto 0);
         overflow: OUT std_logic;
         underflow: OUT std_logic
         );
END div_gen_32_16_float;
--
ARCHITECTURE div_gen_32_16_float_a OF div_gen_32_16_float
IS
-- synopsys translate_off
component wrapped_div_gen_32_16_float
   port (
         clk: IN std_logic;
         ce: IN std_logic;
         aclr: IN std_logic;
         sclr: IN std_logic;
         dividend_mantissa: IN std_logic_VECTOR(31 downto 0);
         dividend_sign: IN std_logic;
         dividend_exponent: IN std_logic_VECTOR(15 downto 0);
         divisor_mantissa: IN std_logic_VECTOR(31 downto 0);
         divisor_sign: IN std_logic;
         divisor_exponent: IN std_logic_VECTOR(15 downto 0);
         quotient_mantissa: OUT std_logic_VECTOR(31 downto 0);
         quotient_sign: OUT std_logic;
         quotient_exponent: OUT std_logic_VECTOR(15 downto 0);
         overflow: OUT std_logic;
         underflow: OUT std_logic
         );
end component;
--
-- Configuration specification
   for all : wrapped_div_gen_32_16_float use entity
XilinxCoreLib.div_gen_v1_0(behavioral)
         generic map(
                        divclk_sel => 1,
                        exponent_width => 16,
                        bias => 0,
                        c_has_sclr => 1,
                        latency => 1,
                        c_has_ce => 1,
                        c_has_aclr => 1,
                        c_sync_enable => 0,
                        fractional_width => 16,
                        mantissa_width => 32,
                        signed_b => 0,
                        fractional_b => 0,
                        algorithm_type => 2,
                        divisor_width => 16,
                        dividend_width => 16
                        );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_div_gen_32_16_float
         port map (
                        clk => clk,
                        ce => ce,
                        aclr => aclr,
                        sclr => sclr,
                        dividend_mantissa => dividend_mantissa,
                        dividend_sign => dividend_sign,
                        dividend_exponent => dividend_exponent,
                        divisor_mantissa => divisor_mantissa,
                        divisor_sign => divisor_sign,
                        divisor_exponent => divisor_exponent,
                        quotient_mantissa => quotient_mantissa,
                        quotient_sign => quotient_sign,
                        quotient_exponent => quotient_exponent,
                        overflow => overflow,
                        underflow => underflow
                        );
-- synopsys translate_on
--
END div_gen_32_16_float_a;
```

Чтобы использовать элемент div_gen_32_16_float в качестве одного из компонентов в составе описания проектируемого устройства, нужно поместить в соответствующий раздел этого описания следующие выражения декларации:

```
component div_gen_32_16_float
   port (
         clk: IN std_logic;
         ce: IN std_logic;
         aclr: IN std_logic;
         sclr: IN std_logic;
         dividend_mantissa: IN std_logic_VECTOR(31 downto 0);
         dividend_sign: IN std_logic;
         dividend_exponent: IN std_logic_VECTOR(15 downto 0);
         divisor_mantissa: IN std_logic_VECTOR(31 downto 0);
         divisor_sign: IN std_logic;
         divisor_exponent: IN std_logic_VECTOR(15 downto 0);
         quotient_mantissa: OUT std_logic_VECTOR(31 downto 0);
         quotient_sign: OUT std_logic;
         quotient_exponent: OUT std_logic_VECTOR(15 downto 0);
         overflow: OUT std_logic;
         underflow: OUT std_logic
         );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of div_gen_32_16_float: component is
«true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of div_gen_32_16_float: component is true;
```

Создание экземпляров компонента div_gen_32_16_float в составе описания архитектуры разрабатываемого устройства осуществляется с помощью оператора, шаблон которого имеет следующий вид:

```
<идентификатор_делителя_div_gen_32_16_float>: div_gen_32_16_float
         port map (
                        clk => clk,
                        ce => ce,
                        aclr => aclr,
                        sclr => sclr,
                        dividend_mantissa => dividend_mantissa,
                        dividend_sign => dividend_sign,
                        dividend_exponent => dividend_exponent,
                        divisor_mantissa => divisor_mantissa,
                        divisor_sign => divisor_sign,
                        divisor_exponent => divisor_exponent,
                        quotient_mantissa => quotient_mantissa,
                        quotient_sign => quotient_sign,
                        quotient_exponent => quotient_exponent,
                        overflow => overflow,
                        underflow => underflow
                        );
```

Пример описания элемента, предназначенного для выполнения операции дробного деления с фиксированной запятой, сформированного на основе параметризированного модуля Divider Generator версии v1.0

Примером элемента, который создан на основе параметризированного модуля *Divider Generator* версии v1.0 с помощью средств CORE Generator для выполнения операции дробного деления с фиксированной запятой, является устройство div_gen_20_20_fract. Этот элемент предназначен для вычисления частного от деления 20-разрядных операндов без знака. Текст сформированного описания элемента div_gen_20_20_fract имеет следующий вид:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY div_gen_20_20_fract IS
   port (
         clk: IN std_logic;
         ce: IN std_logic;
         aclr: IN std_logic;
         sclr: IN std_logic;
```

```
        dividend: IN std_logic_VECTOR(19 downto 0);
        divisor: IN std_logic_VECTOR(19 downto 0);
        quotient: OUT std_logic_VECTOR(19 downto 0);
        remainder: OUT std_logic_VECTOR(19 downto 0);
        rfd: OUT std_logic
        );
END div_gen_20_20_fract;
--
ARCHITECTURE div_gen_20_20_fract_a OF div_gen_20_20_fract IS
-- synopsys translate_off
component wrapped_div_gen_20_20_fract
  port (
        clk: IN std_logic;
        ce: IN std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        dividend: IN std_logic_VECTOR(19 downto 0);
        divisor: IN std_logic_VECTOR(19 downto 0);
        quotient: OUT std_logic_VECTOR(19 downto 0);
        remainder: OUT std_logic_VECTOR(19 downto 0);
        rfd: OUT std_logic
        );
end component;
--
-- Configuration specification
  for all : wrapped_div_gen_20_20_fract use entity
XilinxCoreLib.div_gen_v1_0(behavioral)
        generic map(
                divclk_sel => 1,
                exponent_width => 8,
                bias => 0,
                c_has_sclr => 1,
                latency => 1,
                c_has_ce => 1,
                c_has_aclr => 1,
                c_sync_enable => 1,
                fractional_width => 20,
                mantissa_width => 8,
                signed_b => 0,
                fractional_b => 1,
                algorithm_type => 1,
                divisor_width => 20,
                dividend_width => 20
                );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_div_gen_20_20_fract
        port map (
                clk => clk,
                ce => ce,
                aclr => aclr,
                sclr => sclr,
                dividend => dividend,
                divisor => divisor,
                quotient => quotient,
                remainder => remainder,
                rfd => rfd
                );
-- synopsys translate_on
--
END div_gen_20_20_fract_a;
```

Если элемент div_gen_20_20_fract применяется в качестве компонента в описании разрабатываемого устройства, то в раздел деклараций этого описания необходимо поместить следующую совокупность выражений:

```
component div_gen_20_20_fract
  port (
        clk: IN std_logic;
        ce: IN std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        dividend: IN std_logic_VECTOR(19 downto 0);
        divisor: IN std_logic_VECTOR(19 downto 0);
        quotient: OUT std_logic_VECTOR(19 downto 0);
        remainder: OUT std_logic_VECTOR(19 downto 0);
        rfd: OUT std_logic
        );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of div_gen_20_20_fract: component is
«true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of div_gen_20_20_fract: component is true;
```

Для создания каждого экземпляра компонента div_gen_20_20_fract в описании разрабатываемого устройства следует использовать оператор, шаблон которого выглядит следующим образом:

```
<идентификатор_делителя_div_gen_20_20_fract > : div_gen_20_20_fract
        port map (
                clk => clk,
                ce => ce,
                aclr => aclr,
                sclr => sclr,
                dividend => dividend,
                divisor => divisor,
                quotient => quotient,
                remainder => remainder,
                rfd => rfd
                );
```

Пример описания элемента, предназначенного для выполнения операции целочисленного деления с фиксированной запятой, сформированного на основе параметризированного модуля Divider Generator версии v1.0

VHDL-описание элемента div_gen_16_16_int, представленное в настоящем разделе, демонстрирует результат применения параметризированного модуля *Divider Generator* версии v1.0 для создания устройств, выполняющих операцию целочисленного деления (деления с остатком). Значения делимого и делителя на входных шинах данного элемента представлены в двоичном 16-разрядном коде со знаком. Текст описания элемента div_gen_16_16_int, сгенерированный средствами CORE Generator, имеет следующий вид:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY div_gen_16_16_int IS
  port (
        clk: IN std_logic;
        ce: IN std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        dividend: IN std_logic_VECTOR(15 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quotient: OUT std_logic_VECTOR(15 downto 0);
        remainder: OUT std_logic_VECTOR(15 downto 0);
        rfd: OUT std_logic
        );
END div_gen_16_16_int;
--
ARCHITECTURE div_gen_16_16_int_a OF div_gen_16_16_int IS
-- synopsys translate_off
component wrapped_div_gen_16_16_int
  port (
        clk: IN std_logic;
        ce: IN std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        dividend: IN std_logic_VECTOR(15 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quotient: OUT std_logic_VECTOR(15 downto 0);
        remainder: OUT std_logic_VECTOR(15 downto 0);
        rfd: OUT std_logic
        );
end component;
--
-- Configuration specification
  for all : wrapped_div_gen_16_16_int use entity
XilinxCoreLib.div_gen_v1_0(behavioral)
        generic map(
                divclk_sel => 2,
                exponent_width => 8,
                bias => 0,
                c_has_sclr => 1,
                latency => 1,
                c_has_ce => 1,
                c_has_aclr => 1,
                c_sync_enable => 1,
                fractional_width => 16,
                mantissa_width => 8,


                signed_b => 1,
                fractional_b => 0,
                algorithm_type => 1,
                divisor_width => 16,
                dividend_width => 16
                );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_div_gen_16_16_int
        port map (
                clk => clk,
                ce => ce,
                aclr => aclr,
                sclr => sclr,
                dividend => dividend,
                divisor => divisor,
                quotient => quotient,
                remainder => remainder,
                rfd => rfd
                );
-- synopsys translate_on
--
END div_gen_16_16_int_a;
```

В составе сгенерированного элемента задействованы входы синхронного и асинхронного сброса, а также вход разрешения сигнала синхронизации. Для использования сформированного элемента div_gen_16_16_int в качестве компонента необходимо выполнить его декларацию, поместив в соответствующий раздел VHDL-описания проектируемого устройства приведенную далее последовательность выражений:

```
component div_gen_16_16_int
  port (
        clk: IN std_logic;
        ce: IN std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic;
        dividend: IN std_logic_VECTOR(15 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quotient: OUT std_logic_VECTOR(15 downto 0);
        remainder: OUT std_logic_VECTOR(15 downto 0);
        rfd: OUT std_logic
        );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of div_gen_16_16_int: component is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of div_gen_16_16_int: component is true;
```

Конкретные экземпляры сформированного элемента div_gen_16_16_int, используемые в составе архитектуры разрабатываемого устройства, описываются с помощью оператора, шаблон которого выглядит следующим образом:

```
<идентификатор_делителя_div_gen_16_16_int >: div_gen_16_16_int
        port map (
                clk => clk,
                ce => ce,
                aclr => aclr,
                sclr => sclr,
                dividend => dividend,
                divisor => divisor,
                quotient => quotient,
                remainder => remainder,
                rfd => rfd
                );
```

Продолжение следует