Продолжение. Начало в № 2 ’2007

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

Параметризированный модуль Floating-Point Operator версии v3.0, возможности и характеристики которого были приведены в предыдущей части данной статьи, позволяет формировать описания элементов, предназначенных для выполнения различных операций сравнения значений, представленных в формате с плавающей запятой. В последующих разделах настоящей публикации рассматриваются только три примера описаний указанных элементов, сгенерированных на основе этого ядра, которые соответствуют основным вариантам конфигурации компараторов. К ним относятся: программируемый компаратор, компаратор, формирующий на выходе код состояния, и компаратор с фиксированным типом операции сравнения значений.

Валерий ЗОТОВ
walerry@km.ru

Пример описания программируемого компаратора значений с плавающей запятой, сформированного на основе параметризированного модуля Floating-Point Operator версии v3.0

В качестве примера описания программируемого компаратора значений, представленных в формате с плавающей запятой, которое подготовлено с помощью параметризированного модуля *Floating-Point Operator* версии v3.0, в настоящем разделе приведен текст VHDL-описания элемента *floating_point_compare_programm*. В этом элементе значения сравниваемых входных данных соответствуют стандартному формату чисел с двойной точностью (Double Format). На выходе *result* компаратора *floating_point_compare_programm* формируется сигнал, высокий уровень которого указывает на выполнение соотношения входных значений, определяемого кодом выбранной операции сравнения на шине operation в соответствии с таблицей 3 (см. КиТ № 11 ‘2007).

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY floating_point_compare_programm IS
   port (
         a: IN std_logic_VECTOR(63 downto 0);
         b: IN std_logic_VECTOR(63 downto 0);
         operation: IN std_logic_VECTOR(5 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(0 downto 0);
         invalid_op: OUT std_logic;
         rdy: OUT std_logic
         );
END floating_point_compare_programm;
--
ARCHITECTURE floating_point_compare_programm_a OF floating_point_compare_programm IS
-- synopsys translate_off
component wrapped_floating_point_compare_programm
   port (
         a: IN std_logic_VECTOR(63 downto 0);
         b: IN std_logic_VECTOR(63 downto 0);
         operation: IN std_logic_VECTOR(5 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(0 downto 0);
         invalid_op: OUT std_logic;
         rdy: OUT std_logic
         );
end component;
--
-- Configuration specification
   for all : wrapped_floating_point_compare_programm use entity XilinxCoreLib.floating_point_v3_0(behavioral)
         generic map(
                  c_has_b_nd => 0,
                  c_speed => 2,
                  c_has_sclr => 1,
                  c_has_a_rfd => 0,
                  c_b_fraction_width => 53,
                  c_has_operation_nd => 1,
                  c_family => «virtex4»,
                  c_has_exception => 0,
                  c_a_fraction_width => 53,
                  c_has_flt_to_fix => 0,
                  c_has_flt_to_flt => 0,
                  c_has_fix_to_flt => 0,
                  c_has_invalid_op => 1,
                  c_latency => 3,
                  c_has_divide_by_zero => 0,
                  c_has_overflow => 0,
                  c_mult_usage => 0,
                  c_has_rdy => 1,
                  c_result_fraction_width => 0,
                  c_has_divide => 0,
                  c_has_inexact => 0,
                  c_has_underflow => 0,
                  c_has_sqrt => 0,
                  c_has_add => 0,
                  c_has_status => 0,
                  c_has_a_negate => 0,
                  c_optimization => 1,
                  c_has_a_nd => 0,
                  c_has_aclr => 0,
                  c_has_b_negate => 0,
                  c_has_subtract => 0,
                  c_compare_operation => 8,
                  c_rate => 1,
                  c_has_compare => 1,
                  c_has_operation_rfd => 1,
                  c_has_b_rfd => 0,
                  c_result_width => 1,
                  c_b_width => 64,
                  c_status_early => 0,
                  c_a_width => 64,
                  c_has_cts => 0,
                  c_has_ce => 1,
```

```
                    c_has_multiply => 0
                    );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_floating_point_compare_programm
        port map (
                    a => a,
                    b => b,
                    operation => operation,
                    operation_nd => operation_nd,
                    operation_rfd => operation_rfd,
                    clk => clk,
                    sclr => sclr,
                    ce => ce,
                    result => result,
                    invalid_op => invalid_op,
                    rdy => rdy
                    );
-- synopsys translate_on
--
END floating_point_compare_programm_a;
```

Во всех элементах, выполняющих различные операции сравнения значений с плавающей запятой, которые рассматриваются в данной части статьи (в том числе и в программируемом компараторе *floating_point_compare_program*), задействованы входы синхронного сброса и разрешения синхронизации. В этих компараторах предусмотрена также возможность использования всей совокупности сигналов подтверждения, которые поддерживает параметризированный модуль *Floating-Point Operator* версии v3.0. В состав интерфейса указанных элементов включен выход сигнала, предупреждающего о выполнении недействительной операции.

Для применения программируемого компаратора *floating_point_compare_program* в качестве компонента в описании разрабатываемого устройства нужно поместить в раздел декларации этого описания следующую конструкцию:

```
component floating_point_compare_programm
   port (
        a: IN std_logic_VECTOR(63 downto 0);
        b: IN std_logic_VECTOR(63 downto 0);
        operation: IN std_logic_VECTOR(5 downto 0);
        operation_nd: IN std_logic;
        operation_rfd: OUT std_logic;
        clk: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic;
        result: OUT std_logic_VECTOR(0 downto 0);
        invalid_op: OUT std_logic;
        rdy: OUT std_logic
        );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of floating_point_compare_programm:
component is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of floating_point_compare_programm: com-
ponent is true;
```

Создание конкретных экземпляров сгенерированного программируемого компаратора осуществляется с помощью оператора, шаблон которого выглядит следующим образом:

```
<идентификатор_экземпляра_компаратора_floating_point_
compare_programm> : floating_point_compare_programm
        port map (
                    a => a,
                    b => b,
                    operation => operation,
                    operation_nd => operation_nd,
                    operation_rfd => operation_rfd,
                    clk => clk,
                    sclr => sclr,
                    ce => ce,
                    result => result,
                    invalid_op => invalid_op,
                    rdy => rdy
                    );
```

Сведения о количестве конфигурируемых логических блоков Configurable Logic Block (CLB), таблиц преобразования LookUp Table (LUT) и секций CLB (Slices), используемых для автономной реализации программируемого компаратора *floating_point_compare_program*, отражены в информационной панели, вид которой показан на рис. 88.

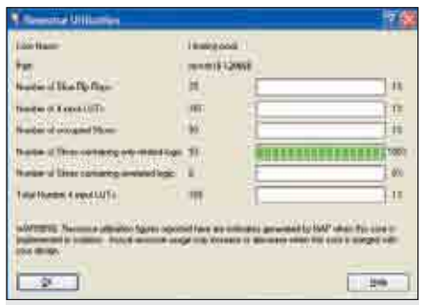

Рис. 88. Вид информационной панели, содержащей сведения о ресурсах ПЛИС, используемых для реализации программируемого компаратора *floating_point_compare_program*

Пример описания компаратора, формирующего на выходе код состояния, значение которого соответствует одному из вариантов соотношения входных данных, представленных в формате с плавающей запятой

Примером компаратора, формирующего на выходе код состояния, значение которого соответствует определенному соотношению входных данных, представленных в формате с плавающей запятой, является описание элемента *floating_point_compare_cc*, сгенерированного на основе параметризированного модуля *Floating-Point Operator* версии v3.0. В разработанном компараторе используется стандартный формат представления значений входных данных с одинарной точностью (*Single Format*). Состояния сигналов на выходах компаратора *floating_point_compare_cc*, соответствующие различным соотношениям значений входных данных, представлены в таблице 5.

Таблица 5. Состояния выходных сигналов компаратора *floating_point_compare_cc* при различных соотношениях значений входных данных

Соотношение значений входных данных	Result(3)	Result(2) >	Result(1) <	Result(0) =
A = B	0	0	0	1
A < B	0	0	1	0
A <= B	0	0	1	1
A > B	0	1	0	0
A >= B	0	1	0	1
A # B	0	1	1	0
A = NaN и/или B = NaN	1	В соответствии со стандартом IEEE Std 754		

Текст сформированного VHDL-описания компаратора *floating_point_compare_cc* выглядит следующим образом:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY floating_point_compare_cc IS
   port (
        a: IN std_logic_VECTOR(31 downto 0);
        b: IN std_logic_VECTOR(31 downto 0);
        operation_nd: IN std_logic;
        operation_rfd: OUT std_logic;
        clk: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic;
        result: OUT std_logic_VECTOR(3 downto 0);
        invalid_op: OUT std_logic;
        rdy: OUT std_logic
        );
END floating_point_compare_cc;
--
ARCHITECTURE floating_point_compare_cc_a OF floating_point_
compare_cc IS
-- synopsys translate_off
component wrapped_floating_point_compare_cc
   port (
        a: IN std_logic_VECTOR(31 downto 0);
        b: IN std_logic_VECTOR(31 downto 0);
        operation_nd: IN std_logic;
        operation_rfd: OUT std_logic;
        clk: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic;
        result: OUT std_logic_VECTOR(3 downto 0);
        invalid_op: OUT std_logic;
        rdy: OUT std_logic
        );
end component;
--
-- Configuration specification
   for all : wrapped_floating_point_compare_cc use entity
XilinxCoreLib.floating_point_v3_0(behavioral)
        generic map(
                    c_has_b_nd => 0,
                    c_speed => 2,
                    c_has_sclr => 1,
                    c_has_a_rfd => 0,
                    c_b_fraction_width => 24,
                    c_has_operation_nd => 1,
                    c_family => «virtex4»,
                    c_has_exception => 0,
                    c_a_fraction_width => 24,
                    c_has_flt_to_fix => 0,
                    c_has_flt_to_flt => 0,
                    c_has_fix_to_flt => 0,
                    c_has_invalid_op => 1,
                    c_latency => 3,
                    c_has_divide_by_zero => 0,
                    c_has_overflow => 0,
                    c_mult_usage => 0,
                    c_has_rdy => 1,
                    c_result_fraction_width => 0,
                    c_has_divide => 0,
                    c_has_inexact => 0,
                    c_has_underflow => 0,
                    c_has_sqrt => 0,
                    c_has_add => 0,
                    c_has_status => 0,
                    c_has_a_negate => 0,
                    c_optimization => 1,
                    c_has_a_nd => 0,
                    c_has_aclr => 0,
                    c_has_b_negate => 0,
                    c_has_subtract => 0,
                    c_compare_operation => 7,
                    c_rate => 1,
                    c_has_compare => 1,
                    c_has_operation_rfd => 1,
                    c_has_b_rfd => 0,
                    c_result_width => 4,
                    c_b_width => 32,
                    c_status_early => 0,
                    c_a_width => 32,
```

```
                    c_has_cts => 0,
                    c_has_ce => 1,
                    c_has_multiply => 0
                    );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_floating_point_compare_cc
        port map (
                    a => a,
                    b => b,
                    operation_nd => operation_nd,
                    operation_rfd => operation_rfd,
                    clk => clk,
                    sclr => sclr,
                    ce => ce,
                    result => result,
                    invalid_op => invalid_op,
                    rdy => rdy
                    );
-- synopsys translate_on
--
END floating_point_compare_cc_a;
```

Выражения декларации компаратора *floating_point_compare_cc* в составе описания разрабатываемого устройства выглядят следующим образом:

```
component floating_point_compare_cc
   port (
        a: IN std_logic_VECTOR(31 downto 0);
        b: IN std_logic_VECTOR(31 downto 0);
        operation_nd: IN std_logic;
        operation_rfd: OUT std_logic;
        clk: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic;
        result: OUT std_logic_VECTOR(3 downto 0);
        invalid_op: OUT std_logic;
        rdy: OUT std_logic);
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of floating_point_compare_cc: component
is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of floating_point_compare_cc: component
is true;
```

Описание конкретных экземпляров компонента *floating_point_compare_cc* осуществляется с помощью оператора, шаблон которого имеет следующий вид:

```
<идентификатор_экземпляра_компаратора_floating_point_
compare_cc>: floating_point_compare_cc
        port map (
                    a => a,
                    b => b,
                    operation_nd => operation_nd,
                    operation_rfd => operation_rfd,
                    clk => clk,
```

Рис. 89. Вид информационной панели, содержащей сведения о ресурсах ПЛИС, используемых для реализации компаратора *floating_point_compare_cc*

```
                    sclr => sclr,
                    ce => ce,
                    result => result,
                    invalid_op => invalid_op,
                    rdy => rdy);
```

На рис. 89 показана информационная панель, которая содержит данные об использовании каждого типа ресурсов ПЛИС в составе сформированного компаратора.

Пример описания компаратора, определяющего эквивалентность значений с плавающей запятой, сформированного на основе параметризированного модуля Floating-Point Operator версии v3.0

Применение параметризированного модуля *Floating-Point Operator* версии v3.0 для подготовки описаний компараторов с фиксированным типом операции сравнения иллюстрируется примером VHDL-описания элемента *floating_point_compare_equal*, определяющего эквивалентность входных значений с плавающей запятой. Значения входных данных в этом компараторе представлены в стандартном формате чисел с двойной точностью (Double Format). Выходной сигнал *result* в рассматриваемом элементе принимает значение высокого логического уровня только при тождественности значений операндов, представленных на его входных шинах. Сформированный текст VHDL-описания компаратора *floating_point_compare_equal* имеет следующий вид:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY floating_point_compare_equal IS
   port (
        a: IN std_logic_VECTOR(63 downto 0);
        b: IN std_logic_VECTOR(63 downto 0);
        operation_nd: IN std_logic;
        operation_rfd: OUT std_logic;
        clk: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic;
        result: OUT std_logic_VECTOR(0 downto 0);
        invalid_op: OUT std_logic;
        rdy: OUT std_logic
        );
END floating_point_compare_equal;
--
ARCHITECTURE floating_point_compare_equal_a OF floating_
point_compare_equal IS
-- synopsys translate_off
component wrapped_floating_point_compare_equal
   port (
        a: IN std_logic_VECTOR(63 downto 0);
        b: IN std_logic_VECTOR(63 downto 0);
        operation_nd: IN std_logic;
        operation_rfd: OUT std_logic;
        clk: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic;
        result: OUT std_logic_VECTOR(0 downto 0);
        invalid_op: OUT std_logic;
        rdy: OUT std_logic
        );
end component;
--
-- Configuration specification
   for all : wrapped_floating_point_compare_equal use entity
XilinxCoreLib.floating_point_v3_0(behavioral)
        generic map(
                    c_has_b_nd => 0,
                    c_speed => 2,
                    c_has_sclr => 1,
                    c_has_a_rfd => 0,
                    c_b_fraction_width => 53,
                    c_has_operation_nd => 1,
                    c_family => «virtex4»,
                    c_has_exception => 0,
                    c_a_fraction_width => 53,
                    c_has_flt_to_fix => 0,
                    c_has_flt_to_flt => 0,
                    c_has_fix_to_flt => 0,
                    c_has_invalid_op => 1,
                    c_latency => 3,
                    c_has_divide_by_zero => 0,
                    c_has_overflow => 0,
                    c_mult_usage => 0,
                    c_has_rdy => 1,
                    c_result_fraction_width => 0,
                    c_has_divide => 0,
                    c_has_inexact => 0,
                    c_has_underflow => 0,
                    c_has_sqrt => 0,
                    c_has_add => 0,
                    c_has_status => 0,
                    c_has_a_negate => 0,
                    c_optimization => 1,
                    c_has_a_nd => 0,
                    c_has_aclr => 0,
                    c_has_b_negate => 0,
                    c_has_subtract => 0,
                    c_compare_operation => 2,
                    c_rate => 1,
                    c_has_compare => 1,
                    c_has_operation_rfd => 1,
                    c_has_b_rfd => 0,
                    c_result_width => 1,
                    c_b_width => 64,
                    c_status_early => 0,
                    c_a_width => 64,
                    c_has_cts => 0,
                    c_has_ce => 1,
                    c_has_multiply => 0
                    );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_floating_point_compare_equal
        port map (
                    a => a,
                    b => b,
                    operation_nd => operation_nd,
                    operation_rfd => operation_rfd,
                    clk => clk,
                    sclr => sclr,
                    ce => ce,
                    result => result,
                    invalid_op => invalid_op,
                    rdy => rdy
                    );
-- synopsys translate_on
--
END floating_point_compare_equal_a;
```

Чтобы использовать данный компаратор в качестве компонента в составе VHDL-описания проектируемого устройства, нужно включить в раздел деклараций архитектурного тела следующую конструкцию:

```
component floating_point_compare_equal
   port (
        a: IN std_logic_VECTOR(63 downto 0);
        b: IN std_logic_VECTOR(63 downto 0);
        operation_nd: IN std_logic;
        operation_rfd: OUT std_logic;
        clk: IN std_logic;
        sclr: IN std_logic;
        ce: IN std_logic;
        result: OUT std_logic_VECTOR(0 downto 0);
        invalid_op: OUT std_logic;
        rdy: OUT std_logic
        );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of floating_point_compare_equal: com-
ponent is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of floating_point_compare_equal: compo-
nent is true;
```

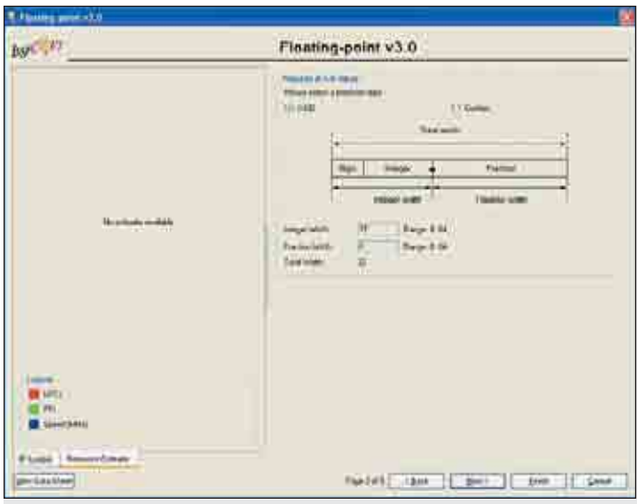

Рис. 91. Диалоговая панель «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, предназначенная для определения точности представления входных данных с фиксированной запятой в формируемом элементе

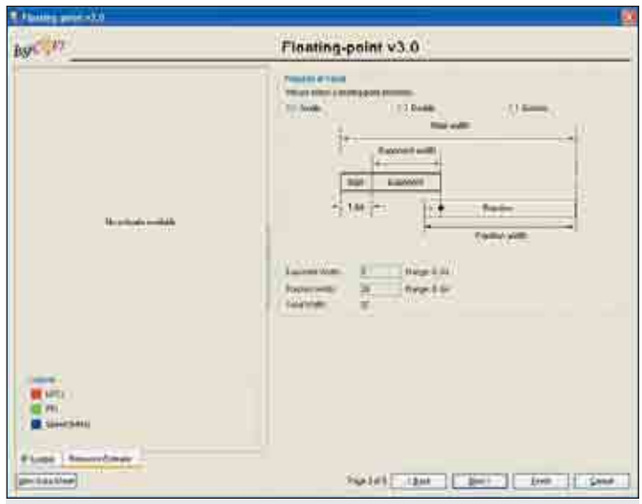

Рис. 92. Диалоговая панель «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, предназначенная для определения точности представления выходных данных с плавающей запятой в формируемом элементе

Для создания конкретных экземпляров компонента *floating_point_compare_equal* в структурном описании разрабатываемого устройства нужно воспользоваться оператором, шаблон которого имеет следующий вид:

```
<идентификатор_экземпляра_компаратора_floating_point_
compare_equal> : floating_point_compare_equal
        port map (
                a => a,
                b => b,
                operation_nd => operation_nd,
                operation_rfd => operation_rfd,
                clk => clk,
                sclr => sclr,
                ce => ce,
                result => result,
                invalid_op => invalid_op,
                rdy => rdy
                );
```

Объем различных ресурсов кристалла, которые необходимы для реализации компаратора *floating_point_compare_equal*, указан в информационной панели, которая представлена на рис. 90.

Формирование описаний элементов, предназначенных для преобразования входных данных из формата с фиксированной запятой в формат с плавающей запятой, на основе параметризированного модуля Floating-Point Operator версии v3.0 с помощью средств CORE Generator

При создании описаний элементов, предназначенных для преобразования формата представления входных данных, в состав «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0 входят не четыре, а пять диалоговых панелей. Содержание этих панелей зависит от выбранного типа операции преобразования формата данных. Определение типа операции преобразования формата значений, выполняемой формируемым элементом, осуществляется с помощью стартовой диалоговой панели этого «мастера», вид которой изображен на рис. 83 в предыдущей части статьи. Чтобы сформировать описание элемента, выполняющего трансляцию входных данных из формата с фиксированной запятой в формат с плавающей запятой, следует во встроенной панели *Operation Selection* зафиксировать в нажатом состоянии кнопку *Fixed-to-float*. После этого нужно перейти ко второй диалоговой панели «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, которая предназначена для определения точности представления входных данных с фиксированной запятой в формируемом элементе. Вид этой диалоговой панели представлен на рис. 91.

С помощью данной диалоговой панели разработчик может выбрать стандартный вариант точности целочисленного представления входных данных или указать произвольные значения длины полей слова данных. Для определения точности представления данных, поступающих на входную шину создаваемого элемента, следует воспользоваться двумя кнопками с зависимой фиксацией, расположенными во встроенной панели *Precision of A/B Inputs* (рис. 91). Если в нажатом состоянии зафиксирована кнопка *Int32*, то входные данные в формируемом элементе будут интерпретироваться как целочисленные 32-разрядные значения. Чтобы выбрать произвольные значения разрядности слова данных и длины его полей, нужно переключить в нажатое положение кнопку *Custom*. При этом станут доступны поля редактирования *Integer Width* и *Fraction Width*, которые расположены в этой же встроенной панели (рис. 91). Длина поля целой части (количество двоичных разрядов, расположенных до запятой) значений входных данных указывается с помощью клавиатуры в поле редактирования *Integer Width*. Требуемое значение длины поля дробной части (количество двоичных разрядов, расположенных после запятой) входных значений задается в поле редактирования *Fraction Width*. Длина каждого из этих полей в слове данных с фиксированной запятой может составлять от нуля до 64 двоичных разрядов. После ввода значений параметров *Integer Width* и *Fraction Width* в строке *Total Width* автоматически отображается значение суммарной длины слова входных данных с фиксированной запятой. Завершив ввод параметров, определяющих точность представления входных значений в формируемом элементе, следует перейти к очередной диалоговой панели «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, вид которой показан на рис. 92.

Рис. 90. Вид информационной панели, содержащей сведения о ресурсах ПЛИС, используемых для реализации компаратора *floating_point_compare_equal*

Данная диалоговая панель предназначена для выбора точности представления значений выходных данных с плавающей запятой (результата преобразования) в формируемом элементе. Выбор требуемой точности представления данных на выходной шине гене-

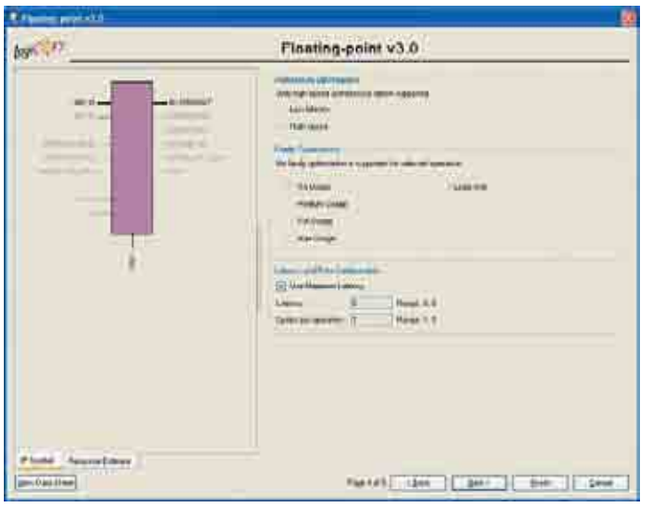

Рис. 93. Диалоговая панель «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, предназначенная для определения временных параметров формируемых элементов

рируемого элемента осуществляется с помощью трех кнопок с зависимой фиксацией, которые представлены во встроенной панели *Precision of Result* (рис. 92). Для формирования выходных данных в стандартном формате представления значений с одинарной точностью (Single Format) нужно переключить в нажатое положение кнопку *Single*. Чтобы подготовить описание элемента, выполняющего преобразование входных данных с фиксированной запятой в стандартный формат значений с двойной точностью (Double Format), следует зафиксировать в нажатом состоянии кнопку *Double*. Если необходимо сгенерировать описание элемента, структура слова данных на выходе которого отличается от стандартных форматов значений с плавающей запятой, нужно установить в нажатое положение кнопку *Custom*. При этом автоматически переключаются в доступное состояние поля редактирования *Exponent Width* и *Fraction Width*, расположенные в этой же встроенной панели (рис. 92). Длина полей показателя степени (порядка) и мантиссы выходного слова данных указывается с помощью клавиатуры в полях редактирования *Exponent Width* и *Fraction Width* соответственно. Задаваемые значения этих параметров должны удовлетворять соотношению (15). Поэтому границы допустимых диапазонов значений, отображаемые справа от указанных полей редактирования, автоматически пересчитываются при каждом изменении одного из них. Значение суммарной длины выходного слова данных, вычисляемое на основании указанной длины полей показателя степени и мантиссы (значений параметров *Exponent Width* и *Fraction Width*), отображается в строке *Total Width*.

После определения точности представления выходных данных в создаваемом элементе нужно перейти к следующей диалоговой панели «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, которая позволяет задать требуемую величину задержки формирования результата. Вид этой диалоговой панели приведен на рис. 93.

Длительность задержки формирования результата в данной диалоговой панели указывается в виде соответствующего числа периодов тактового сигнала. Определение значений этого параметра осуществляется с помощью индикатора состояния и поля редактирования, которые находятся во встроенной панели *Latency and Rate Configuration* (рис. 93). Для формирования элемента, выполняющего операцию преобразования данных из формата с фиксированной запятой в формат с плавающей запятой, с максимально допустимым значением задержки формирования результата нужно установить индикатор *Use Maximum Latency* во включенное состояние. В противном случае значение задержки вычисления результата преобразования входных данных указывается в поле редактирования *Latency*.

Следует обратить внимание на то, что при подготовке описаний элементов, выполняющих операции преобразования формата значений с плавающей запятой, функция предварительной оценки объема используемых ресурсов кристалла является недоступной. В этом случае встроенная панель *Resource Estimates* не содержит никакой информации, как показано на рис. 91, 92.

Завершающим этапом определения параметров формируемого элемента, выполняющего преобразование входных данных из формата с фиксированной запятой в формат с плавающей запятой, является выбор необходимых входов сигналов управления, а также совокупности входов и выходов сигналов подтверждения. Для этих целей используется заключительная диалоговая панель «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, вид которой показан на рис. 87 (он приведен в предыдущей части данной статьи). Выбор входов сигналов управления, включаемых в состав интерфейса создаваемого элемента, производится с помощью индикаторов состояния, которые расположены во встроенной панели *Control Signals*. Для использования в разрабатываемом элементе входа сигнала синхронного сброса необходимо установить в состояние «Включено» индикатор *SCLR*. Чтобы задействовать вход сигнала разрешения синхронизации Clock Enable, нужно перевести во включенное состояние индикатор *CE*.

Состав совокупности входов и выходов сигналов подтверждения, включаемых в состав интерфейса формируемого элемента, определяется с помощью трех индикаторов состояния, представленных во встроенной панели *Handshaking Signals*. Если в создаваемом элементе необходимо задействовать вход сигнала New Data, высокий логический уровень которого указывает на достоверность новых значений данных, представленных на входных шинах, то следует установить в состояние «Включено» индикатор *OPERATION_ND*. Чтобы добавить в состав интерфейса генерируемого устройства выход сигнала Ready for Data, сообщающего о готовности приема и обработки нового слова входных данных, следует переключить в активное состояние индикатор *OPERATION_RFD*. Для присутствия в разрабатываемом элементе выхода сигнала *Ready*, информирующего о готовности (достоверности) результата выполнения операции преобразования входных данных в формат с плавающей запятой, следует установить в состояние «Включено» индикатор *RDY*.

Пример описания элемента, выполняющего операцию преобразования входных значений из формата с фиксированной запятой в формат с плавающей запятой, сформированного на основе параметризированного модуля Floating-Point Operator версии v3.0

В качестве примера описания устройства, сформированного с помощью параметризированного модуля *Floating-Point Operator* версии v3.0 для выполнения операции преобразования входных значений из формата с фиксированной запятой в формат с плавающей запятой, в настоящем разделе приведен текст VHDL-описания элемента *fixed_point_to_floating_point_v3_0*. Данный элемент предназначен для преобразования 54-разрядного слова данных, в котором длина поля дробной части составляет 18 двоичных разрядов, в значение, соответствующее стандартному формату чисел с двойной точностью (Double Format):

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY fixed_point_to_floating_point_v3_0 IS
   port (
         a: IN std_logic_VECTOR(53 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(63 downto 0);
         rdy: OUT std_logic
         );
END fixed_point_to_floating_point_v3_0;
--
ARCHITECTURE fixed_point_to_floating_point_v3_0_a OF
fixed_point_to_floating_point_v3_0 IS
-- synopsys translate_off
component wrapped_fixed_point_to_floating_point_v3_0
   port (
         a: IN std_logic_VECTOR(53 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(63 downto 0);
         rdy: OUT std_logic
         );
end component;
--
-- Configuration specification
   for all : wrapped_fixed_point_to_floating_point_v3_0 use entity
XilinxCoreLib.floating_point_v3_0(behavioral)
         generic map(
                        c_has_b_nd => 0,
                        c_speed => 2,
                        c_has_sclr => 1,
                        c_has_a_rfd => 0,
                        c_b_fraction_width => 18,
                        c_has_operation_nd => 1,
                        c_family => «virtex4»,
                        c_has_exception => 0,
                        c_a_fraction_width => 18,
                        c_has_flt_to_fix => 0,
                        c_has_flt_to_flt => 0,
                        c_has_fix_to_flt => 1,
                        c_has_invalid_op => 0,
                        c_latency => 0,
                        c_has_divide_by_zero => 0,
                        c_has_overflow => 0,
                        c_mult_usage => 0,
                        c_has_rdy => 1,
                        c_result_fraction_width => 53,
                        c_has_divide => 0,
                        c_has_inexact => 0,
                        c_has_underflow => 0,
                        c_has_sqrt => 0,
                        c_has_add => 0,
                        c_has_status => 0,
                        c_has_a_negate => 0,
                        c_optimization => 1,
                        c_has_a_nd => 0,
                        c_has_aclr => 0,
                        c_has_b_negate => 0,
                        c_has_subtract => 0,
                        c_compare_operation => 8,
                        c_rate => 1,
                        c_has_compare => 0,
                        c_has_operation_rfd => 1,
                        c_has_b_rfd => 0,
                        c_result_width => 64,
                        c_b_width => 54,
                        c_status_early => 0,
                        c_a_width => 54,
                        c_has_cts => 0,
                        c_has_ce => 1,
                        c_has_multiply => 0
                        );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_fixed_point_to_floating_point_v3_0
         port map (
                        a => a,
                        operation_nd => operation_nd,
                        operation_rfd => operation_rfd,
                        clk => clk,
                        sclr => sclr,
                        ce => ce,
                        result => result,
                        rdy => rdy
                        );
-- synopsys translate_on
--
END fixed_point_to_floating_point_v3_0_a;
```

В сформированном элементе *fixed_point_to_floating_point_v3_0* задействованы входы синхронного сброса и вход разрешения синхронизации. Кроме того, в этом элементе присутствуют все входы и выходы сигналов подтверждения, поддерживаемых параметризированным модулем *Floating-Point Operator* версии v3.0.

При использовании представленного элемента *fixed_point_to_floating_point_v3_0* в качестве компонента проектируемого устройства нужно поместить в состав соответствующего раздела формируемого VHDL-описания этого устройства следующие выражения декларации:

```
component fixed_point_to_floating_point_v3_0
   port (
         a: IN std_logic_VECTOR(53 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(63 downto 0);
         rdy: OUT std_logic
         );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of fixed_point_to_floating_point_v3_0:
component is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of fixed_point_to_floating_point_v3_0: com-
ponent is true;
```

Чтобы добавить экземпляр компонента *fixed_point_to_floating_point_v3_0* в описание разрабатываемого устройства, необходимо включить в состав этого описания оператор, шаблон которого имеет следующий вид:

```
<идентификатор_экземпляра_элемента_fixed_point_to_floating_
point_v3_0> : fixed_point_to_floating_point_v3_0
         port map (
                        a => a,
                        operation_nd => operation_nd,
                        operation_rfd => operation_rfd,
                        clk => clk,
                        sclr => sclr,
                        ce => ce,
                        result => result,
                        rdy => rdy
                        );
```

Подробные сведения о количестве ресурсов кристалла, требуемых для реализации элемента *fixed_point_to_floating_point_v3_0*, демонстрирует информационная панель, представленная на рис. 94.

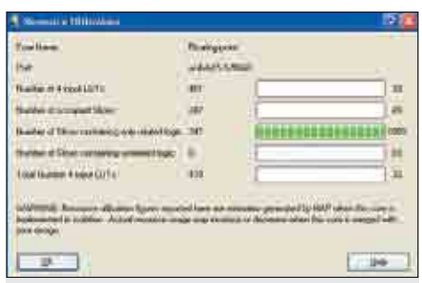

Рис. 94. Вид информационной панели, содержащей сведения о ресурсах ПЛИС, необходимых для реализации элемента *fixed_point_to_floating_point_v3_0*

Подготовка описаний элементов, предназначенных для преобразования входных данных из формата с плавающей запятой в формат с фиксированной запятой, на основе параметризированного модуля Floating-Point Operator версии v3.0 с помощью средств CORE Generator

Для создания описания элемента, выполняющего преобразование входных данных из формата с плавающей запятой в формат с фиксированной запятой, необходимо во встроенной панели *Operation Selection*, расположенной в стартовой диалоговой панели «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, установить в нажатое состояние кнопку *Float-to-fixed*. Дальнейший процесс определения параметров генерируемого устройства включает в себя ту же последовательность этапов, что и при разработке описаний элементов, предназначенных для преобразования входных данных из формата с фиксированной запятой в формат с плавающей запятой. Поэтому в настоящем разделе рассматриваются только особенности этого процесса.

Вторая диалоговая панель «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, предназначенная для определения точности представления входных данных с плавающей запятой в формируемом элементе, приобретает вид, показанный на рис. 95. С помощью кнопок с зависимой фиксацией *Single*, *Double*, *Custom* и полей редактирования *Exponent Width*, *Fraction Width*, расположенных в этой диалоговой панели, выбирается длина полей показателя степени и мантиссы во входном слове данных, причем таким же образом, как и аналогичные параметры представления выходных данных (результата) в элементах, выполняющих операцию преобразования значений из формата с фиксированной запятой в формат с плавающей запятой.

Изменяется также вид третьей диалоговой панели «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, которая используется для выбора точности представления выходных данных с фиксированной запятой в формируемом элементе. Вид этой диалоговой панели изображен на рис. 96.

В заключительной диалоговой панели «мастера» настройки, кроме выбора входов сигналов управления (синхронного сброса и разрешения синхронизации) и выходов сигналов подтверждения, разработчику предоставляется возможность использования в формируемом элементе выходов некоторых специальных сигналов, информирующих о возникно-

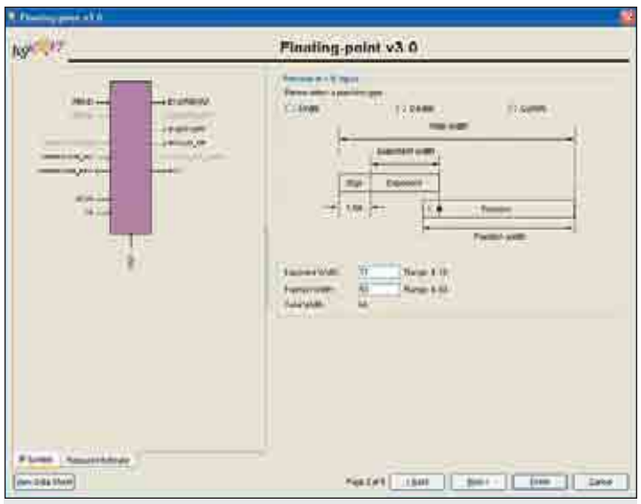

Рис. 95. Вид второй диалоговой панели «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, предназначенной для определения точности представления входных данных с плавающей запятой в формируемом элементе

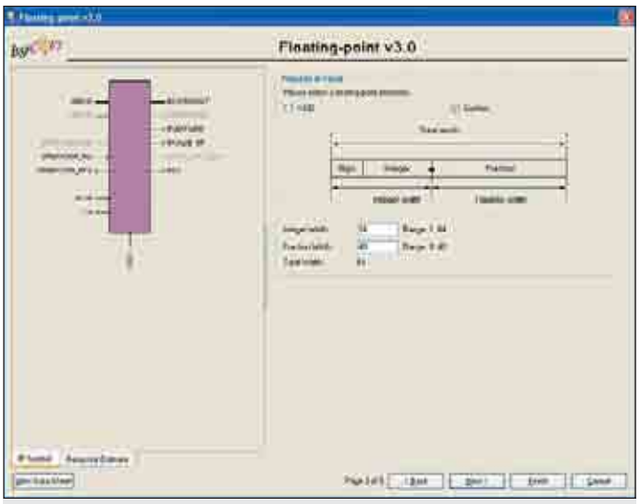

Рис. 96. Вид третьей диалоговой панели «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0, предназначенной для выбора точности представления выходных данных с фиксированной запятой в формируемом элементе

вении различных исключительных ситуаций при выполнении операций преобразования формата входных данных. Чтобы задействовать в создаваемом элементе выход сигнала переполнения, необходимо установить в состояние «Включено» индикатор *OVERFLOW*. Для использования в составе разрабатываемого элемента выхода сигнала, информирующего о выполнении недействительной операции, следует перевести во включенное состояние индикатор *INVALID_OPERATION*.

Пример описания элемента, выполняющего операцию преобразования входных значений из формата с плавающей запятой в формат с фиксированной запятой, сформированного на основе параметризированного модуля Floating-Point Operator версии v3.0

Примером описания устройства, сгенерированного для преобразования входных данных из формата с плавающей запятой в формат с фиксированной запятой на основе параметризированного модуля *Floating-Point Operator* версии v3.0 с помощью средств CORE Generator, является VHDL-описание элемента *floating_point_to_fixed_point_v3_0*, текст которого представлен в настоящем разделе. Данный элемент выполняет трансляцию 42-разрядных значений с плавающей запятой, поступающих на входную шину данных, в 64-разрядное слово данных с фиксированной запятой. Полученный результат преобразования входных данных отображается на выходной шине элемента *floating_point_to_fixed_point_v3_0* в виде 64-разрядного значения, в котором длина полей целой и дробной части составляет 32 разряда:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY floating_point_to_fixed_point_v3_0 IS
   port (
         a: IN std_logic_VECTOR(41 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(63 downto 0);
         overflow: OUT std_logic;
         invalid_op: OUT std_logic;
         rdy: OUT std_logic
         );
END floating_point_to_fixed_point_v3_0;
--
ARCHITECTURE floating_point_to_fixed_point_v3_0_a OF floating_point_to_fixed_point_v3_0 IS
-- synopsys translate_off
component wrapped_floating_point_to_fixed_point_v3_0
   port (
         a: IN std_logic_VECTOR(41 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(63 downto 0);
         overflow: OUT std_logic;
         invalid_op: OUT std_logic;
         rdy: OUT std_logic
         );
end component;
--
-- Configuration specification
   for all : wrapped_floating_point_to_fixed_point_v3_0 use entity XilinxCoreLib.floating_point_v3_0(behavioral)
         generic map(
                  c_has_b_nd => 0,
                  c_speed => 2,
                  c_has_sclr => 1,
                  c_has_a_rfd => 0,
                  c_b_fraction_width => 32,
                  c_has_operation_nd => 1,
                  c_family => «virtex4»,
                  c_has_exception => 0,
                  c_a_fraction_width => 32,
                  c_has_flt_to_fix => 1,
                  c_has_flt_to_flt => 0,
                  c_has_fix_to_flt => 0,
                  c_has_invalid_op => 1,
                  c_latency => 5,
                  c_has_divide_by_zero => 0,
                  c_has_overflow => 1,
                  c_mult_usage => 0,
                  c_has_rdy => 1,
                  c_result_fraction_width => 32,
                  c_has_divide => 0,
                  c_has_inexact => 0,
                  c_has_underflow => 0,
                  c_has_sqrt => 0,
                  c_has_add => 0,
                  c_has_status => 0,
                  c_has_a_negate => 0,
                  c_optimization => 1,
                  c_has_a_nd => 0,
                  c_has_aclr => 0,
                  c_has_b_negate => 0,
                  c_has_subtract => 0,
                  c_compare_operation => 8,
                  c_rate => 1,
                  c_has_compare => 0,
                  c_has_operation_rfd => 1,
                  c_has_b_rfd => 0,
                  c_result_width => 64,
                  c_b_width => 42,
                  c_status_early => 0,
                  c_a_width => 42,
                  c_has_cts => 0,
                  c_has_ce => 1,
                  c_has_multiply => 0
                  );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_floating_point_to_fixed_point_v3_0
         port map (
                  a => a,
                  operation_nd => operation_nd,
                  operation_rfd => operation_rfd,
                  clk => clk,
                  sclr => sclr,
                  ce => ce,
                  result => result,
                  overflow => overflow,
                  invalid_op => invalid_op,
                  rdy => rdy
                  );
-- synopsys translate_on
--
END floating_point_to_fixed_point_v3_0_a;
```

В сформированном элементе *floating_point_to_fixed_point_v3_0* задействованы те же входы сигналов управления и выходы сигналов подтверждения, что и в элементе *fixed_point_to_floating_point_v3_0*, описание которого было представлено ранее. Кроме того, в составе разработанного элемента, выполняющего преобразование значений из формата с плавающей запятой в формат с фиксированной запятой, присутствуют выходы сигнала переполнения и признака недействительной операции.

Для применения элемента *floating_point_to_fixed_point_v3_0* в качестве компонента проектируемого устройства следует включить в состав раздела декларации VHDL-описания этого устройства приведенную далее конструкцию:

```
component floating_point_to_fixed_point_v3_0
   port (
         a: IN std_logic_VECTOR(41 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(63 downto 0);
         overflow: OUT std_logic;
         invalid_op: OUT std_logic;
         rdy: OUT std_logic
         );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of floating_point_to_fixed_point_v3_0:
component is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of floating_point_to_fixed_point_v3_0: component is true;
```

Создание конкретных экземпляров компонента *floating_point_to_fixed_point_v3_0* в составе описания архитектуры разрабатываемого устройства осуществляется с помощью оператора, шаблон которого выглядит следующим образом:

```
<идентификатор_экземпляра_элемента_floating_point_to_fixed_
point_v3_0>: floating_point_to_fixed_point_v3_0
         port map (
                  a => a,
                  operation_nd => operation_nd,
                  operation_rfd => operation_rfd,
                  clk => clk,
                  sclr => sclr,
                  ce => ce,
                  result => result,
                  overflow => overflow,
                  invalid_op => invalid_op,
                  rdy => rdy
                  );
```

На рис. 97 представлена информационная панель, в которой содержатся данные об использовании различных ресурсов ПЛИС для автономной реализации элемента *floating_point_to_fixed_point_v3_0*.

Рис. 97. Вид информационной панели, содержащей сведения о ресурсах кристалла, используемых для реализации элемента *floating_point_to_fixed_point_v3_0*

Создание описаний элементов, предназначенных для преобразования входных данных из одного формата с плавающей запятой в другой формат с плавающей запятой, на основе параметризированного модуля Floating-Point Operator версии v3.0 с помощью средств CORE Generator

Процесс подготовки описаний элементов, предназначенных для преобразования входных данных из одного формата с плавающей запятой в другой формат с плавающей запятой, начинается с переключения в нажатое положение кнопки *Float-to-float* во встроенной панели *Operation Selection*, находящейся в стартовой диалоговой панели «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0. Дальнейшая последовательность этапов выполнения этого процесса, осуществляемая с помощью «мастера» настройки, включает в себя те же действия, что и при создании описаний элементов, реализующих операцию преобразования входных данных из формата с фиксированной запятой в формат с плавающей запятой. Отличия проявляются в структуре некоторых диалоговых панелей «мастера» настройки параметров ядра *Floating-Point Operator* версии v3.0.

Определение точности представления входных значений в генерируемом элементе производится с помощью второй диалоговой панели «мастера» настройки, которая имеет вид, показанный на рис. 95. Параметры структуры выходного слова данных указываются в третьей диалоговой панели, вид которой изображен на рис. 92. Длина полей показателя степени и мантиссы входных и выходных значений с плавающей запятой выбирается теми же способами, что и при формировании описаний элементов, выполняющих операцию преобразования входных данных из формата с плавающей запятой в формат с фиксированной запятой.

Заключительная диалоговая панель «мастера» настройки позволяет включить в состав интерфейса создаваемого элемента, предназначенного для преобразования входных данных из одного формата с плавающей запятой в другой формат с плавающей запятой, все входы сигналов управления и выходы сигналов подтверждения, поддерживаемые параметризированным модулем *Floating-Point Operator* версии v3.0. Кроме того, данная диалоговая панель предоставляет разработчику возможность использования в формируемых элементах выходов сигналов, информирующих о переполнении или исчезновении значащих разрядов в результате выполнения операции преобразования формата входных данных. Для включения в состав интерфейса генерируемого элемента выхода сигнала переполнения необходимо установить в состояние «Включено» индикатор *OVERFLOW*, который находится во встроенной панели *Exception Signals*. Если в создаваемом элементе должен присутствовать выход сигнала, уведомляющего об исчезновении значащих разрядов в полученном результате преобразования, то следует переключить в активное состояние индикатор *UNDERFLOW*, расположенный в этой же встроенной панели.

Пример описания элемента, выполняющего операцию преобразования входных значений из одного формата с плавающей запятой в другой формат с плавающей запятой, сформированного на основе параметризированного модуля Floating-Point Operator версии v3.0

Результат применения параметризированного модуля *Floating-Point Operator* версии v3.0 для генерации устройств, реализующих операцию преобразования входных значений из одного формата с плавающей запятой в другой формат с плавающей запятой, демонстрируется на примере VHDL-описания элемента *floating_point_sf_to_floating_point_df_v3_0*, текст которого приведен далее в настоящем разделе. Этот элемент предназначен для трансляции входного слова данных, представленных в стандартном формате с одинарной точностью (*Single Format*), в значение, соответствующее стандартному формату чисел с двойной точностью (Double Format):

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on
ENTITY floating_point_sf_to_floating_point_df_v3_0 IS
   port (
         a: IN std_logic_VECTOR(31 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(63 downto 0);
         underflow: OUT std_logic;
         overflow: OUT std_logic;
         rdy: OUT std_logic
         );
END floating_point_sf_to_floating_point_df_v3_0;
--
ARCHITECTURE floating_point_sf_to_floating_point_df_v3_0_a
OF floating_point_sf_to_floating_point_df_v3_0 IS
-- synopsys translate_off
component wrapped_floating_point_sf_to_floating_point_df_v3_0
   port (
         a: IN std_logic_VECTOR(31 downto 0);
         operation_nd: IN std_logic;
         operation_rfd: OUT std_logic;
         clk: IN std_logic;
         sclr: IN std_logic;
         ce: IN std_logic;
         result: OUT std_logic_VECTOR(63 downto 0);
         underflow: OUT std_logic;
         overflow: OUT std_logic;
         rdy: OUT std_logic
         );
end component;
--
-- Configuration specification
   for all : wrapped_floating_point_sf_to_floating_point_df_v3_0 use
entity XilinxCoreLib.floating_point_v3_0(behavioral)
         generic map(
                  c_has_b_nd => 0,
                  c_speed => 2,
                  c_has_sclr => 1,
                  c_has_a_rfd => 0,
```

```
                    c_b_fraction_width => 24,
                    c_has_operation_nd => 1,
                    c_family => «virtex4»,
                    c_has_exception => 0,
                    c_a_fraction_width => 24,
                    c_has_flt_to_fix => 0,
                    c_has_flt_to_flt => 1,
                    c_has_fix_to_flt => 0,
                    c_has_invalid_op => 0,
                    c_latency => 2,
                    c_has_divide_by_zero => 0,
                    c_has_overflow => 1,
                    c_mult_usage => 0,
                    c_has_rdy => 1,
                    c_result_fraction_width => 53,
                    c_has_divide => 0,
                    c_has_inexact => 0,
                    c_has_underflow => 1,
                    c_has_sqrt => 0,
                    c_has_add => 0,
                    c_has_status => 0,
                    c_has_a_negate => 0,
                    c_optimization => 1,
                    c_has_a_nd => 0,
                    c_has_aclr => 0,
                    c_has_b_negate => 0,
                    c_has_subtract => 0,
                    c_compare_operation => 8,
                    c_rate => 1,
                    c_has_compare => 0,
                    c_has_operation_rfd => 1,
                    c_has_b_rfd => 0,
                    c_result_width => 64,
                    c_b_width => 32,
                    c_status_early => 0,
                    c_a_width => 32,
                    c_has_cts => 0,
                    c_has_ce => 1,
                    c_has_multiply => 0
                    );
-- synopsys translate_on
BEGIN
-- synopsys translate_off
U0 : wrapped_floating_point_sf_to_floating_point_df_v3_0
        port map (
                    a => a,
                    operation_nd => operation_nd,
                    operation_rfd => operation_rfd,
                    clk => clk,
                    sclr => sclr,
                    ce => ce,
                    result => result,
                    underflow => underflow,
                    overflow => overflow,
                    rdy => rdy
                    );
-- synopsys translate_on
--
END floating_point_sf_to_floating_point_df_v3_0_a;
```

Если элемент *floating_point_sf_to_floating_point_df_v3_0* применяется в качестве одного из компонентов VHDL-описания проектируемого устройства, то его декларация выполняется с помощью следующей совокупности выражений:

```
component floating_point_sf_to_floating_point_df_v3_0
    port (
            a: IN std_logic_VECTOR(31 downto 0);
            operation_nd: IN std_logic;
            operation_rfd: OUT std_logic;
            clk: IN std_logic;
            sclr: IN std_logic;
            ce: IN std_logic;
            result: OUT std_logic_VECTOR(63 downto 0);
            underflow: OUT std_logic;
            overflow: OUT std_logic;
            rdy: OUT std_logic
            );
end component;
--
-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of floating_point_sf_to_floating_point_df_
v3_0: component is «true»;
--
-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of floating_point_sf_to_floating_point_df_
v3_0: component is true;
```

Конкретные экземпляры компонента *floating_point_sf_to_floating_point_df_v3_0* в блоке определения архитектуры VHDL-описания разрабатываемого устройства создаются с помощью оператора, шаблон которого выглядит следующим образом:

```
<идентификатор_экземпляра_элемента_ floating_point_sf_to_floating_
point_df_v3_0>: floating_point_sf_to_floating_point_df_v3_0
        port map (
                    a => a,
                    operation_nd => operation_nd,
                    operation_rfd => operation_rfd,
                    clk => clk,
                    sclr => sclr,
                    ce => ce,
                    result => result,
                    underflow => underflow,
                    overflow => overflow,
                    rdy => rdy
                    );
```

В информационной панели, приведенной на рис. 98, содержатся данные о количестве различных ресурсов кристалла, используемых для реализации элемента *floating_point_sf_to_floating_point_df_v3_0*.

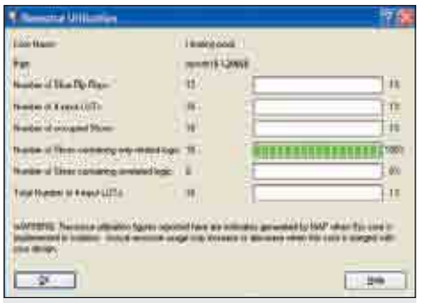

Рис. 98. Вид информационной панели, содержащей сведения о ресурсах ПЛИС, необходимых для реализации элемента *floating_point_sf_to_floating_point_df_v3_0*

Продолжение следует