Использование приложения HDL Coder системы MATLAB/Simulink для реализации квантованных КИХ-фильтров в базисе ПЛИС

Андрей СТРОГОНОВ,
д. т. н.
andreistrogonov@mail.ru
Алексей БЫСТРИЦКИЙ,
к. т. н.
bystritsky@bk.ru

Использование формата с фиксированной запятой позволяет получать высокую скорость вычислений [1]. Однако при проектировании квантованных КИХ-фильтров с фиксированной запятой необходимо учитывать следующие факторы: диапазон для результатов вычислений; требуемую погрешность результата; ошибки, связанные с квантованием; алгоритм реализации вычислений и др. Поэтому цель авторов статьи — показать, как можно осуществить переход от имитационной модели КИХ-фильтра, разработанной в системе MATLAB/Simulink, к функциональной, реализованной в САПР ПЛИС Quartus II компании Altera, с учетом квантования.

Числа с фиксированной запятой характеризуются длиной слова в битах, положением двоичной точки (binary point) и могут быть беззнаковыми или знаковыми. Позиция двоичной точки определяет число разрядов в целой и дробной частях машинного слова. Для представления знаковых чисел (отрицательных и положительных) старший разряд двоичного слова отводится под знак числа (sign bit). При представлении беззнаковых чисел с фиксированной точкой разряд знака отсутствует, и он становится значимым разрядом. Отрицательные числа представляются в дополнительном коде.

Данные с фиксированной запятой могут быть следующих типов:

- целыми (integers);
- дробными (fractional);
- обобщенными (generalize) [2].

В данных обобщенного типа нет возможности определить позицию двоичной запятой по умолчанию, и поэтому необходимо явно указать ее положение. Этот тип данных специфицируют форматами ufix и sfix.

На рис. 1 представлено двоичное число с фиксированной запятой обобщенного типа, где b_i — i-й разряд числа; n — длина двоичного слова в битах; b_{n-1} — старший значимый разряд (MSB); b_0 — младший значимый разряд (LSB); 2^i — вес i-го разряда числа. Двоичная запятая занимает четвертую позицию от младшего (LSB) разряда числа. При этом длина дробной части числа $m = 4$.

В файле помощи Fixed-Point Blockset [3, 4] для представления такого числа применяется следующая формула:

$$V = S \times Q + B,$$

где V — точное значение действительного десятичного числа; S — наклон; Q — квантованное (двоично-взвешенное) значение целого числа; B — смещение.

Наклон S представляется следующим образом:

$$S = F \times 2^E,$$

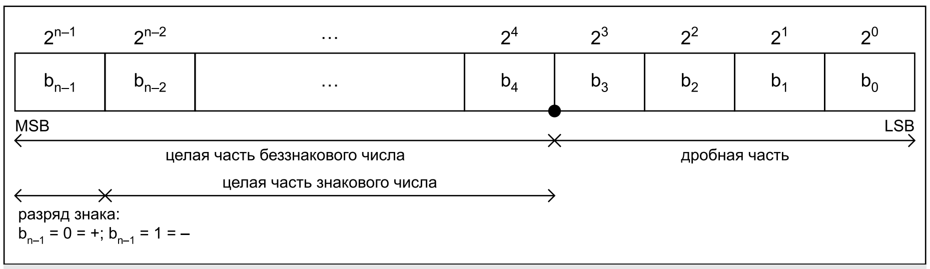

Рис. 1. Формат машинного слова

где F — наклон дробной части, нормализованная величина $1 \le F < 2$; E — показатель степени $E = -m$. При проектировании устройств цифровой обработки сигналов принимают $B = 0$ и $F = 1$:

$$V \approx 2^{-m} \times Q.$$

Квантованное значение Q приближенно представляет истинное значение действительного числа V в виде суммы произведений весовых коэффициентов b_i на вес 2^i соответствующих двоичных разрядов машинного слова.

Для беззнаковых чисел с фиксированной точкой Q определяется формулой:

$$Q = \sum_{i=0}^{n-1} b_i \times 2^i.$$

Квантованное значение знаковых чисел определяется по формуле:

$$Q = -b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i.$$

Так как целые числа не имеют дробной части ($m = 0$), то выражение для V имеет вид:

$$V = \sum_{i=0}^{n-1} b_i \times 2^i$$

и для знакового целого числа:

$$V = -b_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} b_i \times 2^i.$$

В формате с фиксированной запятой без знака вещественное число V можно считать обозначением полинома:

$$V = S\times\left[\sum_{i=0}^{n-1} b_i 2^i\right].$$

Например, двоичное число в дополнительном коде 0011.0101 при длине машинного слова $n = 8$ и $m = 4$ представляет беззнаковое (MSB = 0) вещественное число 3,3125:

$$3{,}3125 = 2^{-4}(0\times2^7+0\times2^6+1\times2^5+1\times2^4+ +0\times2^3+1\times2^2+0\times2^1+1\times2^0).$$

При MSB = 1 будем иметь уже другое число: –4,6875:

$$-4{,}6875 = 2^{-4}(1\times2^7+0\times2^6+1\times2^5+1\times2^4+ +0\times2^3+1\times2^2+0\times2^1+1\times2^0).$$

Рассмотрим два варианта извлечения кода языка VHDL из моделей расширения Simulink (среда моделирования систем непрерывного и дискретного времени) системы MATLAB. Существует еще вариант извлечения кода непосредственно из алгоритмов MATLAB. В первом варианте код из описания структуры фильтра извлекается базовыми элементами расширения Simulink, а во втором — с помощью языка M-файлов Simulink.

На рис. 2 показана имитационная модель (верхний уровень иерархии) симметричного КИХ-фильтра на восемь отводов, взятая из демонстрационного примера Simulink HDL Coder Examples Symmetric FIR Filters [5]. На вход фильтра поступает зашумленный сигнал (всего 2001 отсчетов):

```
x_in = cos(2.*pi.*(0:0.001:2).*(1+(0:0.001:2).*75)).'.
```

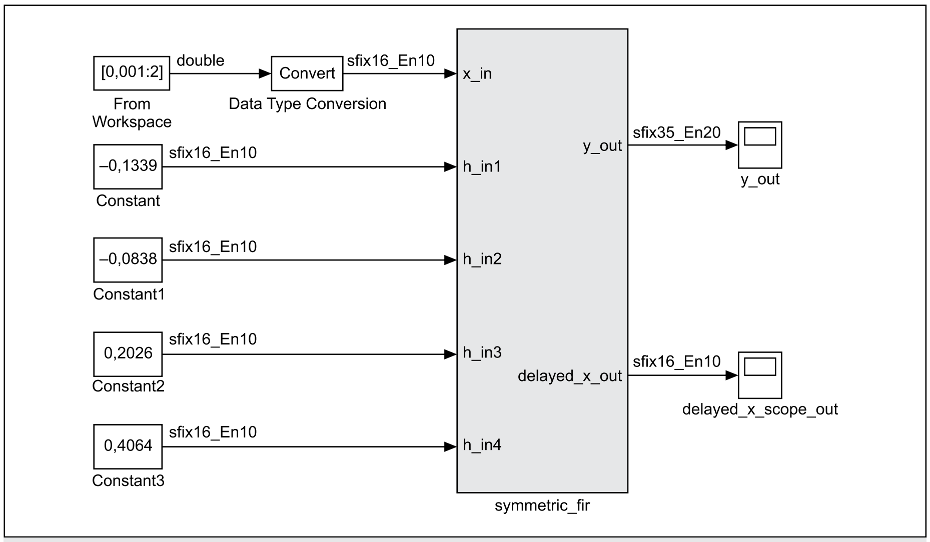

Рис. 2. Имитационная модель симметричного КИХ-фильтра на восемь отводов (верхний уровень иерархии)

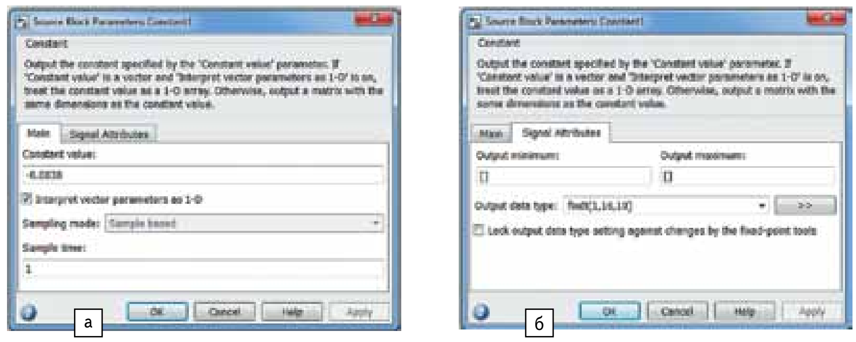

Рис. 3. Настройка функционального блока Constant с именем Constant1:
а) задается вещественное число –0,083, являющееся одним из коэффициентов фильтра (закладка Main);
б) выходной формат представления этого числа — fixdt (1, 16, 10) (закладка Signal Attributes)

Предположим, что коэффициенты фильтра известны и хранятся в функциональных блоках Constant с такими же именами, например Constant3. Выходные сигналы представляются в формате с фиксированной запятой fixdt (1, 16, 10):

- h1 = –0,1339;
- h2 = –0,0838;
- h3 = 0,2026;
- h4 = 0,4064.

Преобразовывать значения из формата с плавающей запятой в формат с фиксированной запятой позволяют «автоматизированные мастера», встроенные в функциональные блоки (рис. 3).

Коэффициенты фильтра подвергаются масштабированию путем умножения на масштабный множитель 1024 в соответствии с выбранным форматом 16.10 и последующему округлению. (Например, h1×1024 = –137,1136 округляется до целого значения –137D или ff77 в дополнительном коде при длине машинного слова 16 бит.) В шестнадцатеричной системе счисления с учетом знака числа они будут выглядеть следующим образом: ff77, ffaa, 00cf, 01a0. При этом следует помнить, что формат квантования коэффициентов КИХ-фильтра влияет на его частотные и временные характеристики.

Значения входного сигнала, который необходимо профильтровать, изменяются по амплитуде от –1 до +1. Поэтому, если они представлены в формате с фиксированной запятой, их необходимо разделить на 1024. Например, начальные значения сигнала выглядят следующим образом: 0400, 03ff, 03ff, 03ff, 03ff, 03ff, 03fe, или поделенные на 1024: 1, 0,9990234375 и т. д.

Функциональный блок Data Tupe Conversion осуществляет преобразование формата double в fixdt (1, 16, 10).

Рассмотрим имитационную модель КИХ-фильтра на рис. 2. После запуска модели над входным сигналом x_in, коэффициентами фильтра и выходными сигналами y_out и delayed_x_scope_out (выход линии задержки) указывается используемый формат.

При использовании арифметики с фиксированной запятой операции сложения не приводят к необходимости округления результатов: они могут лишь вызвать переполнение разрядной сетки. Поэтому выходы с пресумматоров Add–Add3 представляются в формате sfix17_En10 для учета переполнения. Выходы с умножителей Product–Product3 представляются в формате sfix33_En20, так как умножение чисел с фиксированной запятой приводит к увеличению числа значащих цифр результата по сравнению с сомножителями, которые в данном случае 16- и 17-разрядные, а значит, к необходимости округления. Выходы с сумматоров Add5–Add6 первого уровня дерева сумматоров представляются в формате sfix34_En20, а второго уровня — в формате sfix35_En20. Следовательно, результат фильтрации (сигнал y_out) представляется в формате sfix35_En20 (рис. 4).

Далее с помощью приложения Simulink HDL Coder системы MATLAB/Simulink извлечем в автоматическом режиме код языка VHDL КИХ-фильтра (пример 1). Запуск приложения осуществляется из меню **Code → HDL Code → Generate HDL**. Предварительно с помощью меню **Code → HDL Code → Option** необходимо заказать определенные опции и осуществить настройки.

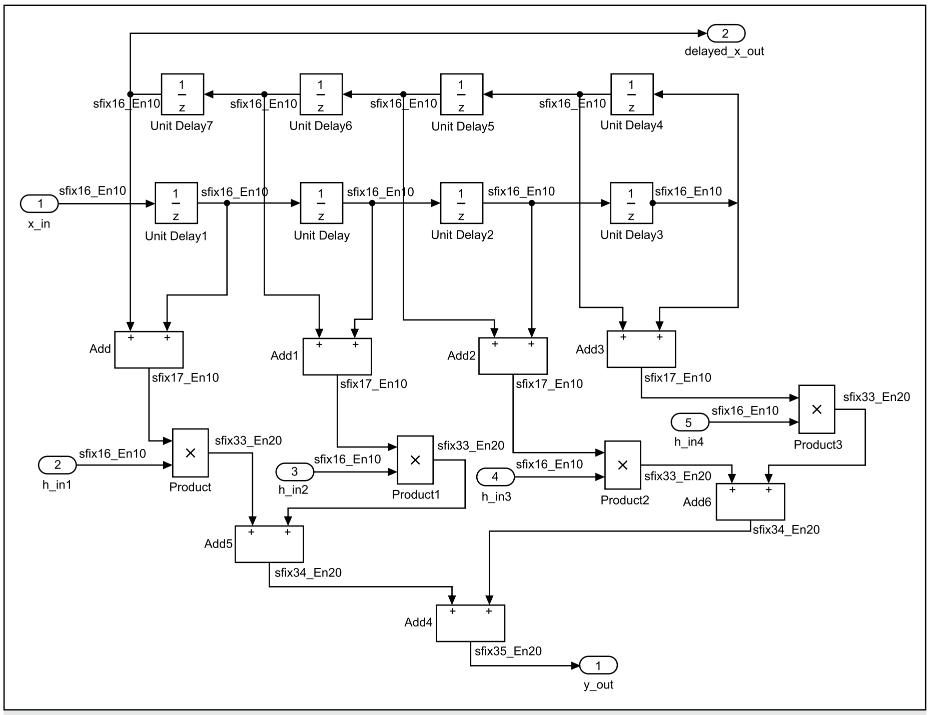

Рис. 4. Имитационная модель симметричного КИХ-фильтра на восемь отводов с указанием формата сигналов (нижний уровень иерархии; структура фильтра в элементах системы MATLAB/Simulink)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ENTITY symmetric_fir IS
  PORT( clk IN std_logic;
    reset : IN std_logic;
    clk_enable : IN std_logic;
      x_in : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      h_in1 : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      h_In2 : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      h_In3 : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      h_In4 : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
    ce_out : OUT std_logic;
      y_out : OUT std_logic_vector(34 DOWNTO 0); -- sfix35_En20
      delayed_x_out : OUT std_logic_vector(15 DOWNTO 0) -- sfix16_
En10
    );
END symmetric_fir;
ARCHITECTURE rtl OF symmetric_fir IS
  -- Signals
  SIGNAL enb : std_logic;
  SIGNAL x_in_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Unit_Delay1_out1 : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Unit_Delay_out1 : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Unit_Delay2_out1 : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Unit_Delay3_out1 : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Unit_Delay4_out1 : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Unit_Delay5_out1 : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Unit_Delay6_out1 : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Unit_Delay7_out1 : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Add_add_cast : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add_add_cast_1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add_out1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add1_add_cast : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add1_add_cast_1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add1_out1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add2_add_cast : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add2_add_cast_1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add2_out1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add3_add_cast : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add3_add_cast_1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL Add3_out1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL h_in1_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Product_out1 : signed(32 DOWNTO 0); -- sfix33_En20
  SIGNAL h_In2_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Product1_out1 : signed(32 DOWNTO 0); -- sfix33_En20
  SIGNAL Add5_add_cast : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL Add5_add_cast_1 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL Add5_out1 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL h_In3_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Product2_out1 : signed(32 DOWNTO 0); -- sfix33_En20
  SIGNAL h_In4_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL Product3_out1 : signed(32 DOWNTO 0); -- sfix33_En20
  SIGNAL Add6_add_cast : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL Add6_add_cast_1 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL Add6_out1 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL Add4_add_cast : signed(34 DOWNTO 0); -- sfix35_En20
  SIGNAL Add4_add_cast_1 : signed(34 DOWNTO 0); -- sfix35_En20
  SIGNAL Add4_out1 : signed(34 DOWNTO 0); -- sfix35_En20
BEGIN
  x_in_signed <= signed(x_in);
  enb <= clk_enable;
  -- <S1>/Unit Delay1
  Unit_Delay1_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay1_out1 <= to_signed(0, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay1_out1 <= x_in_signed;
      END IF;
    END IF;
  END PROCESS Unit_Delay1_process;
  -- <S1>/Unit Delay
  Unit_Delay_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay_out1 <= to_signed(0, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay_out1 <= Unit_Delay1_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay_process;
  -- <S1>/Unit Delay2
  Unit_Delay2_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay2_out1 <= to_signed(0, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay2_out1 <= Unit_Delay_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay2_process;
  -- <S1>/Unit Delay3
  Unit_Delay3_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay3_out1 <= to_signed(0, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay3_out1 <= Unit_Delay2_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay3_process;
  -- <S1>/Unit Delay4
  Unit_Delay4_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay4_out1 <= to_signed(0, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay4_out1 <= Unit_Delay3_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay4_process;
  -- <S1>/Unit Delay5
  Unit_Delay5_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay5_out1 <= to_signed(0, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay5_out1 <= Unit_Delay4_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay5_process;
  -- <S1>/Unit Delay6
  Unit_Delay6_process : PROCESS (clk, reset)
  BEGIN
  IF reset = '1' THEN
    Unit_Delay6_out1 <= to_signed(0, 16);
  ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay6_out1 <= Unit_Delay5_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay6_process;
  -- <S1>/Unit Delay7
  Unit_Delay7_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay7_out1 <= to_signed(0, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay7_out1 <= Unit_Delay6_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay7_process;
  -- <S1>/Add
  Add_add_cast <= resize(Unit_Delay7_out1, 17);
  Add_add_cast_1 <= resize(Unit_Delay1_out1, 17);
  Add_out1 <= Add_add_cast + Add_add_cast_1;
  -- <S1>/Add1
  Add1_add_cast <= resize(Unit_Delay6_out1, 17);
  Add1_add_cast_1 <= resize(Unit_Delay_out1, 17);
  Add1_out1 <= Add1_add_cast + Add1_add_cast_1;
  -- <S1>/Add2
  Add2_add_cast <= resize(Unit_Delay5_out1, 17);
  Add2_add_cast_1 <= resize(Unit_Delay2_out1, 17);
  Add2_out1 <= Add2_add_cast + Add2_add_cast_1;
  -- <S1>/Add3
  Add3_add_cast <= resize(Unit_Delay4_out1, 17);
  Add3_add_cast_1 <= resize(Unit_Delay3_out1, 17);
  Add3_out1 <= Add3_add_cast + Add3_add_cast_1;
  h_in1_signed <= signed(h_in1);
  -- <S1>/Product
  Product_out1 <= Add_out1 * h_in1_signed;
  h_In2_signed <= signed(h_In2);
  -- <S1>/Product1
  Product1_out1 <= Add1_out1 * h_In2_signed;
  -- <S1>/Add5
  Add5_add_cast <= resize(Product_out1, 34);
  Add5_add_cast_1 <= resize(Product1_out1, 34);
  Add5_out1 <= Add5_add_cast + Add5_add_cast_1;
  h_In3_signed <= signed(h_In3);
  -- <S1>/Product2
  Product2_out1 <= Add2_out1 * h_In3_signed;
  h_In4_signed <= signed(h_In4);
  -- <S1>/Product3
  Product3_out1 <= Add3_out1 * h_In4_signed;
  -- <S1>/Add6
  Add6_add_cast <= resize(Product2_out1, 34);
  Add6_add_cast_1 <= resize(Product3_out1, 34);
  Add6_out1 <= Add6_add_cast + Add6_add_cast_1;
  -- <S1>/Add4
  Add4_add_cast <= resize(Add5_out1, 35);
  Add4_add_cast_1 <= resize(Add6_out1, 35);
  Add4_out1 <= Add4_add_cast + Add4_add_cast_1;
  y_out <= std_logic_vector(Add4_out1);
  delayed_x_out <= std_logic_vector(Unit_Delay7_out1);
  ce_out <= clk_enable;
END rtl;
```

Пример 1. Код языка VHDL, извлеченный в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов

Рассмотрим второй вариант. Разработаем имитационную модель симметричного КИХ-фильтра на восемь отводов в формате с фиксированной запятой, с помощью fi-объектов

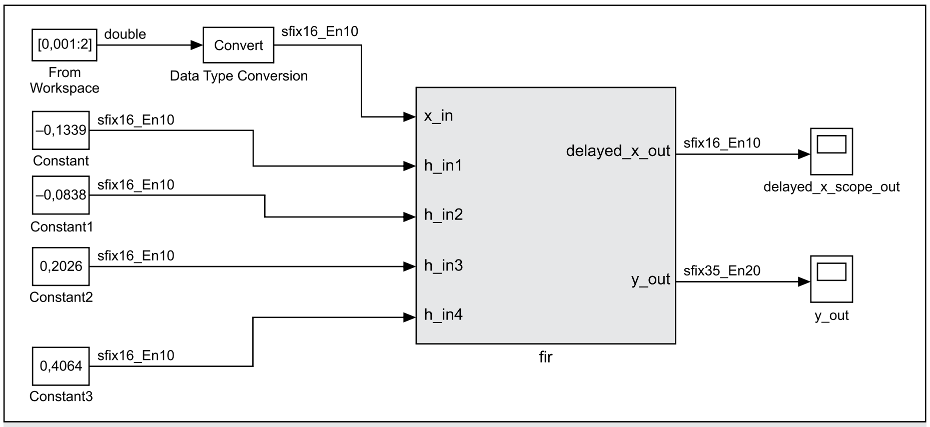

Рис. 5. Верхний уровень иерархии имитационной модели симметричного КИХ-фильтра на восемь отводов с использованием М-файла

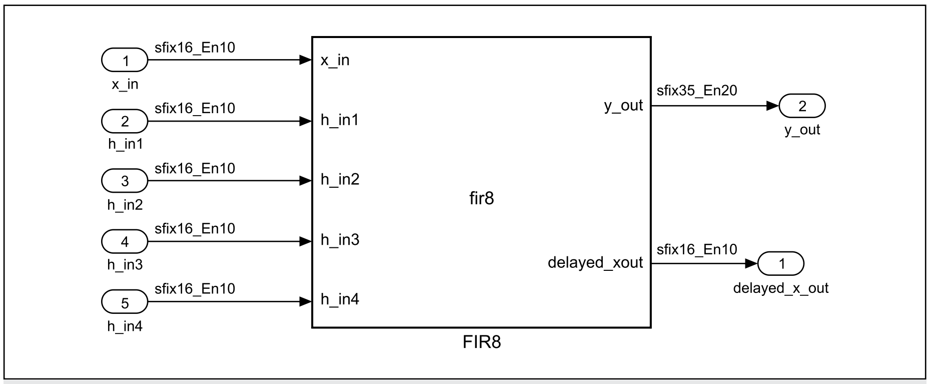

Рис. 6. Подсистема симметричного КИХ-фильтра на восемь отводов (подключение входных и выходных портов к М-файлу)

и языка М-файлов системы MATLAB (рис. 5, 6). Используем следующий формат для представления десятичных чисел:

```
a = fi(v, s, w, f),
```

где v — десятичное число; s — знак 0 (false) для чисел без знака и 1 (true) для чисел со знаком; w — размер слова в битах (целая часть числа); f — дробная часть числа в битах.

Это можно осуществить на основе следующего формата:

```
a = fi(v, s, w, f, fimath).
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
```

Такие настройки вычислений в формате с фиксированной запятой приняты в системе Simulink по умолчанию. Можно задать режим округления (RoundMode) — floor — округление вниз; реакцию на переполнение (OverflowMode) — wrap — перенос. При выходе значения v из допустимого диапазона «лишние» старшие разряды игнорируются. При выполнении операций умножения (ProductMode) и сложения (SumMode) для повышения точности вычислений (precision) используется машинное слово шириной в 32 бита.

Во примере 2 показан М-файл симметричного КИХ-фильтра на восемь отводов с использованием fi-объектов. А на рис. 7 представлен сигнал до и после фильтрации.

```
%#codegen
function [y_out, delayed_xout] = fir8(x_in, h_in1, h_in2, h_in3, h_in4)
% Symmetric FIR Filter
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
% declare and initialize the delay registers
persistent ud1 ud2 ud3 ud4 ud5 ud6 ud7 ud8;
if isempty(ud1)
    ud1 = fi(0, 1, 16, 10, hdl_fm);
    ud2 = fi(0, 1, 16, 10, hdl_fm);
    ud3 = fi(0, 1, 16, 10, hdl_fm);
    ud4 = fi(0, 1, 16, 10, hdl_fm);
    ud5 = fi(0, 1, 16, 10, hdl_fm);
    ud6 = fi(0, 1, 16, 10, hdl_fm);
    ud7 = fi(0, 1, 16, 10, hdl_fm);
    ud8 = fi(0, 1, 16, 10, hdl_fm);
end
% access the previous value of states/registers
a1 = fi(ud1 + ud8, 1, 17, 10, hdl_fm);
a2 = fi(ud2 + ud7, 1, 17, 10, hdl_fm);
a3 = fi(ud3 + ud6, 1, 17, 10, hdl_fm);
a4 = fi(ud4 + ud5, 1, 17, 10, hdl_fm);
% multiplier chain
m1 = fi((h_in1*a1), 1, 33, 20, hdl_fm);
m2 = fi((h_in2*a2), 1, 33, 20, hdl_fm);
m3 = fi((h_in3*a3), 1, 33, 20, hdl_fm);
m4 = fi((h_in4*a4), 1, 33, 20, hdl_fm);
% adder chain
a5 = fi(m1 + m2, 1, 34, 20, hdl_fm);
a6 = fi(m3 + m4, 1, 34, 20, hdl_fm);
% filtered output
y_out = fi(a5 + a6, 1, 35, 20, hdl_fm);
% delayout input signal
delayed_xout = ud8;
% update the delay line
ud8 = ud7;
ud7 = ud6;
ud6 = ud5;
ud5 = ud4;
ud4 = ud3;
ud3 = ud2;
ud2 = ud1;
ud1 = fi(x_in, 1, 16, 10, hdl_fm);
end
```

Пример 2. М-файл симметричного КИХ-фильтра на восемь отводов с использованием fi-объектов

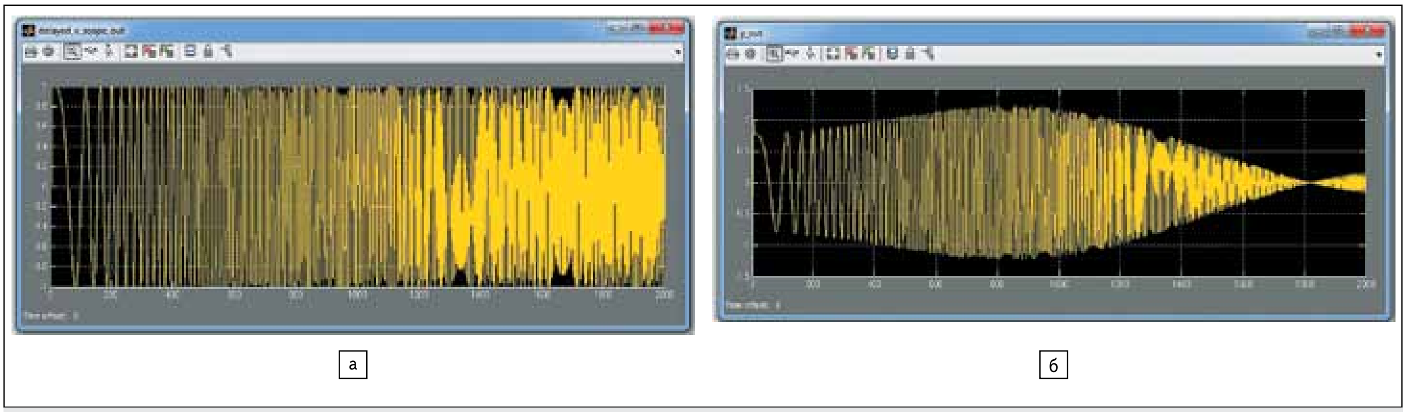

Рис. 7. Результаты имитационного моделирования: а) исходный сигнал; б) профильтрованный сигнал

Пример 3 демонстрирует код языка VHDL, извлеченный в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов.

```
-- File Name: C:\fir\fir8_vhdl\FIR8.vhd
-- Created: 2014-03-04 15:01:23
-- Generated by MATLAB 8.0 and HDL Coder 3.1
-- Module: FIR8
-- Source Path: fir_new/fir/FIR8
-- Hierarchy Level: 1
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ENTITY FIR8 IS
  PORT( clk : IN std_logic;
    reset : IN std_logic;
    enb : IN std_logic;
    x_in : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      h_in1 : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      h_in2 : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      h_in3 : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      h_in4 : IN std_logic_vector(15 DOWNTO 0); -- sfix16_En10
      y_out : OUT std_logic_vector(34 DOWNTO 0); -- sfix35_En20
      delayed_xout : OUT std_logic_vector(15 DOWNTO 0) -- sfix16_En10
    );
END FIR8;
ARCHITECTURE rtl OF FIR8 IS
  -- Signals
  SIGNAL x_in_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL h_in1_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL h_in2_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL h_in3_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL h_in4_signed : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL y_out_tmp : signed(34 DOWNTO 0); -- sfix35_En20
  SIGNAL delayed_xout_tmp : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL ud1 : signed(15 DOWNTO 0); -- sfix16
  SIGNAL ud2 : signed(15 DOWNTO 0); -- sfix16
  SIGNAL ud3 : signed(15 DOWNTO 0); -- sfix16
  SIGNAL ud4 : signed(15 DOWNTO 0); -- sfix16
  SIGNAL ud5 : signed(15 DOWNTO 0); -- sfix16
  SIGNAL ud6 : signed(15 DOWNTO 0); -- sfix16
  SIGNAL ud7 : signed(15 DOWNTO 0); -- sfix16
  SIGNAL ud8 : signed(15 DOWNTO 0); -- sfix16
  SIGNAL ud2_next : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL ud3_next : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL ud4_next : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL ud5_next : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL ud6_next : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL ud7_next : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL ud8_next : signed(15 DOWNTO 0); -- sfix16_En10
  SIGNAL a1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL a2 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL a3 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL a4 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL m1 : signed(32 DOWNTO 0); -- sfix33_En20
  SIGNAL m2 : signed(32 DOWNTO 0); -- sfix33_En20
  SIGNAL m3 : signed(32 DOWNTO 0); -- sfix33_En20
  SIGNAL m4 : signed(32 DOWNTO 0); -- sfix33_En20
  SIGNAL a5 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL a6 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL add_cast : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL add_cast_1 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL add_cast_2 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL add_cast_3 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL add_cast_4 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL add_cast_5 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL add_cast_6 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL add_cast_7 : signed(16 DOWNTO 0); -- sfix17_En10
  SIGNAL add_cast_8 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL add_cast_9 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL add_cast_10 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL add_cast_11 : signed(33 DOWNTO 0); -- sfix34_En20
  SIGNAL add_cast_12 : signed(34 DOWNTO 0); -- sfix35_En20
  SIGNAL add_cast_13 : signed(34 DOWNTO 0); -- sfix35_En20
BEGIN
  x_in_signed <= signed(x_in);
  h_in1_signed <= signed(h_in1);
  h_in2_signed <= signed(h_in2);
  h_in3_signed <= signed(h_in3);
  h_in4_signed <= signed(h_in4);
  FIR8_1_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      ud1 <= to_signed(0, 16);
      ud2 <= to_signed(0, 16);
      ud3 <= to_signed(0, 16);
      ud4 <= to_signed(0, 16);
      ud5 <= to_signed(0, 16);
      ud6 <= to_signed(0, 16);
      ud7 <= to_signed(0, 16);
      ud8 <= to_signed(0, 16);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        ud2 <= ud2_next;
        ud3 <= ud3_next;
        ud4 <= ud4_next;
        ud5 <= ud5_next;
        ud6 <= ud6_next;
        ud7 <= ud7_next;
        ud8 <= ud8_next;
        ud1 <= x_in_signed;
      END IF;
    END IF;
  END PROCESS FIR8_1_process;
  --MATLAB Function 'fir/FIR8': '<S2>:1'
  -- Symmetric FIR Filter
  -- HDL specific fimath
  -- declare and initialize the delay registers
  -- access the previous value of states/registers
  --'<S2>:1:27'
  add_cast <= resize(ud1, 17);
  add_cast_1 <= resize(ud8, 17);
  a1 <= add_cast + add_cast_1;
  --'<S2>:1:28'
  add_cast_2 <= resize(ud2, 17);
  add_cast_3 <= resize(ud7, 17);
  a2 <= add_cast_2 + add_cast_3;
  --'<S2>:1:29'
  add_cast_4 <= resize(ud3, 17);
  add_cast_5 <= resize(ud6, 17);
  a3 <= add_cast_4 + add_cast_5;
  --'<S2>:1:30'
  add_cast_6 <= resize(ud4, 17);
  add_cast_7 <= resize(ud5, 17);
  a4 <= add_cast_6 + add_cast_7;
  -- multiplier chain
  --'<S2>:1:33'
  m1 <= h_in1_signed * a1;
  --'<S2>:1:34'
  m2 <= h_in2_signed * a2;
  --'<S2>:1:35'
  m3 <= h_in3_signed * a3;
  --'<S2>:1:36'
  m4 <= h_in4_signed * a4;
  -- adder chain
  --'<S2>:1:39'
  add_cast_8 <= resize(m1, 34);
  add_cast_9 <= resize(m2, 34);
  a5 <= add_cast_8 + add_cast_9;
  --'<S2>:1:40'
  add_cast_10 <= resize(m3, 34);
  add_cast_11 <= resize(m4, 34);
  a6 <= add_cast_10 + add_cast_11;
  -- filtered output
  --'<S2>:1:43'
  add_cast_12 <= resize(a5, 35);
  add_cast_13 <= resize(a6, 35);
  y_out_tmp <= add_cast_12 + add_cast_13;
  -- delayout input signal
  --'<S2>:1:46'
  delayed_xout_tmp <= ud8;
  -- update the delay line
  --'<S2>:1:49'
  ud8_next <= ud7;
  --'<S2>:1:50'
  ud7_next <= ud6;
  --'<S2>:1:51'
  ud6_next <= ud5;
  --'<S2>:1:52'
  ud5_next <= ud4;
  --'<S2>:1:53'
  ud4_next <= ud3;
  --'<S2>:1:54'
  ud3_next <= ud2;
  --'<S2>:1:55'
  ud2_next <= ud1;
  --'<S2>:1:56'
  y_out <= std_logic_vector(y_out_tmp);
  delayed_xout <= std_logic_vector(delayed_xout_tmp);
END rtl;
```

Пример 3. Код языка VHDL, извлеченный в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;
USE work.fir_tb_pkg.ALL;
PACKAGE fir_tb_data IS
  CONSTANT Data_Type_Conversion_out1_force : Data_Type_
Conversion_out1_type;
  CONSTANT Constant_out1_force : std_logic_vector(15 DOWNTO 0);
  CONSTANT Constant1_out1_force : std_logic_vector(15 DOWNTO 0);
  CONSTANT Constant2_out1_force : std_logic_vector(15 DOWNTO 0);
  CONSTANT Constant3_out1_force : std_logic_vector(15 DOWNTO 0);
  CONSTANT delayed_x_out_expected : Data_Type_Conversion_out1_type;
  CONSTANT y_out_expected : y_out_type;
END fir_tb_data;
PACKAGE BODY fir_tb_data IS
-- Входной сигнал
  CONSTANT Data_Type_Conversion_out1_force : Data_Type_
Conversion_out1_type :=
    (
      X"0400",
      X"03ff",
      X"03ff",
      X"03ff",
      X"03ff",
      X"03ff",
      X"03fe",
      X"03fd",
      X"03fc",
      X"03fb",
      X"03f9",
      X"03f7",
      X"03f5",
      X"03f2",
      X"03ef",
      X"03eb",
-- Коэффициенты фильтра
  CONSTANT Constant_out1_force : std_logic_vector(15 DOWNTO 0) :=
    (
      X"ff77");
  CONSTANT Constant1_out1_force : std_logic_vector(15 DOWNTO 0) :=
    (
      X"ffaa");
  CONSTANT Constant2_out1_force : std_logic_vector(15 DOWNTO 0) :=
    (
      X"00cf");
 CONSTANT Constant3_out1_force : std_logic_vector(15 DOWNTO 0) :=
    (
      X"01a0");
  CONSTANT delayed_x_out_expected : Data_Type_Conversion_
out1_type :=
    (
      X"0000",
      X"0000",
      X"0000",
      X"0000",
      X"0000",
      X"0000",
      X"0000",
      X"0000",
      X"0400",
      X"03ff",
      X"03ff",
      X"03ff",
-- Отклик фильтра, профильтрованные значения
CONSTANT y_out_expected : y_out_type :=
    (
      SLICE(X"000000000",35),
      SLICE(X"7fffddc00",35),
      SLICE(X"7fffc8489",35),
      SLICE(X"7ffffc0df",35),
      SLICE(X"000064010",35),
      SLICE(X"0000cbe70",35),
      SLICE(X"0000ff8d0",35),
      SLICE(X"0000ea08a",35),
      SLICE(X"0000c7dbf",35),
      SLICE(X"0000c7e58",35),
      SLICE(X"0000c7cc8",35),
      SLICE(X"0000c7a21",35),
      SLICE(X"0000c7701",35),
      SLICE(X"0000c7368",35),
      SLICE(X"0000c6f41",35),
      SLICE(X"0000c69d0",35),
      SLICE(X"0000c634a",35),
      SLICE(X"0000c5bd0",35)
```

Пример 4. Фрагмент тестбенча, полученный с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов

Рассмотрим функциональное моделирование КИХ-фильтра с использованием симулятора ModelSim-Altera Starter Edition. На рис. 8 показано моделирование фильтра с использованием тестбенча (пример 4), полученного с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов.

Используя полученный код (пример 3), разработаем функциональную модель в САПР ПЛИС Quartus II (рис. 9). Входной сигнал сформируем с помощью векторного редактора (рис. 10) в соответствии с примером 4. Проект размещен в ПЛИС EP2C5AF256A7 серии Cyclone II. На рис. 11 показано распределение задействованных ресурсов проекта по кристаллу ПЛИС EP2C5AF256A7. Используется восемь аппаратных умножителей с размерностью операндов 9×9. Это равносильно использованию четырех умножителей с размерностью операндов 18×18.

Таблица. Общие сведения по числу задействованных ресурсов ПЛИС Cyclone II EP2C5AF256A7

Общее число логических элементов (Logic Cells)	Количество триггеров логических элементов (Dedicated logic registers)	Количество аппаратных умножителей размерностью операндов 9×9 (Embedded Multiplier 9-bit elements)	Рабочая частота в наихудшем случае F_{max}, МГц
238/4608 (5%)	128/4608 (3%)	8/26 (31%) или четыре умножителя с размерностью 18×18	380

Сравнивая рис. 8 и 10, видим, что функциональная модель КИХ-фильтра на восемь отводов, построенная с использованием кода языка VHDL, извлеченного в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов в САПР ПЛИС Quartus II версии 13.0, работает корректно.

В состав HDL Coder входит инструмент под названием HDL Workflow Advisor (помощник по работе с HDL), который автоматизирует программирование ПЛИС фирм Xilinx и Altera. Можно контролировать HDL-архитектуру и реализацию, выделять критические пути и генерировать отчеты об использовании аппаратных ресурсов.

На рис. 12 показан программный инструмент HDL Workflow Advisor расширения Simulink. Более подробную информацию можно получить на сайте matlab.ru [5]. Программный инструмент HDL Workflow Advisor дублирует пункты меню **Code → HDL Code → Generate HDL** и **Code → HDL Code → Option**, но позволяет работать на более качественном уровне.

Генерация кода происходит в несколько этапов. Первый этап заключается в выборе стиля проектирования: реализация проекта осуществляется в базисе заказных БИС/ПЛИС без привязки к конкретному производителю или при выборе отладочной платы и серии ПЛИС фирм Xilinx или Altera. Для проектов в базисе ПЛИС Workflow Advisor обеспечивает такие возможности оптимизации, как площадь-скорость, распределение памяти RAM, конвейеризация, совместное использование ресурсов и развертывание циклов.

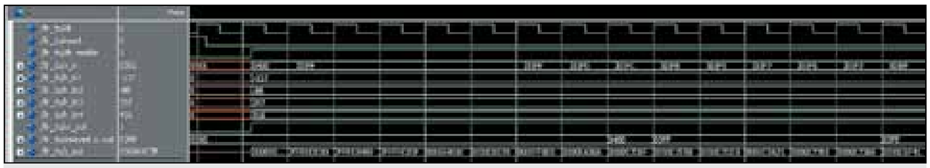

Рис. 8. Функциональное моделирование фильтра с использованием тестбенча, полученного с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов

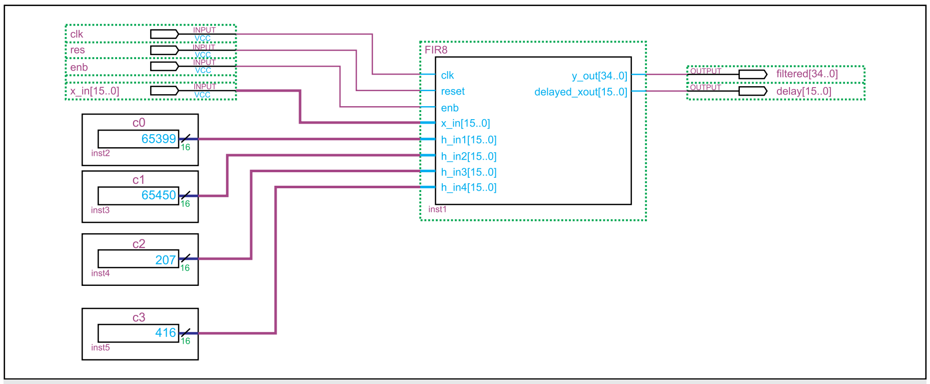

Рис. 9. Функциональная модель КИХ-фильтра на восемь отводов, построенная с использованием кода языка VHDL, извлеченного в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов в САПР ПЛИС Quartus II версии 13.0

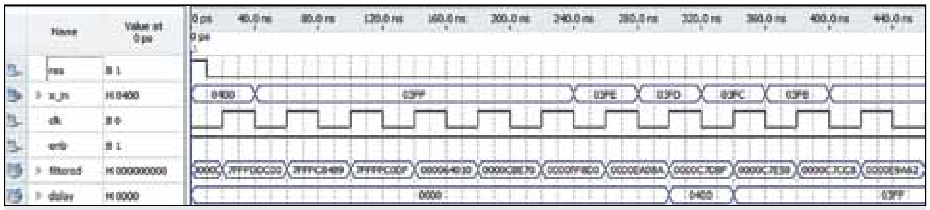

Рис. 10. Функциональное моделирование фильтра с использованием кода языка VHDL, извлеченного в автоматическом режиме с помощью приложения Simulink HDL Coder из имитационной модели симметричного КИХ-фильтра на восемь отводов на основе M-файлов и fi-объектов

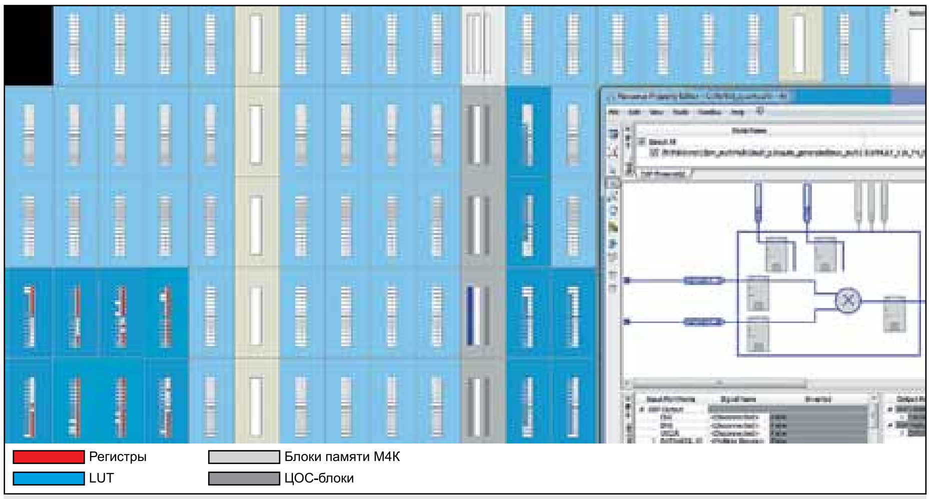

Рис. 11. Распределение задействованных ресурсов проекта по кристаллу ПЛИС EP2C5AF256A7

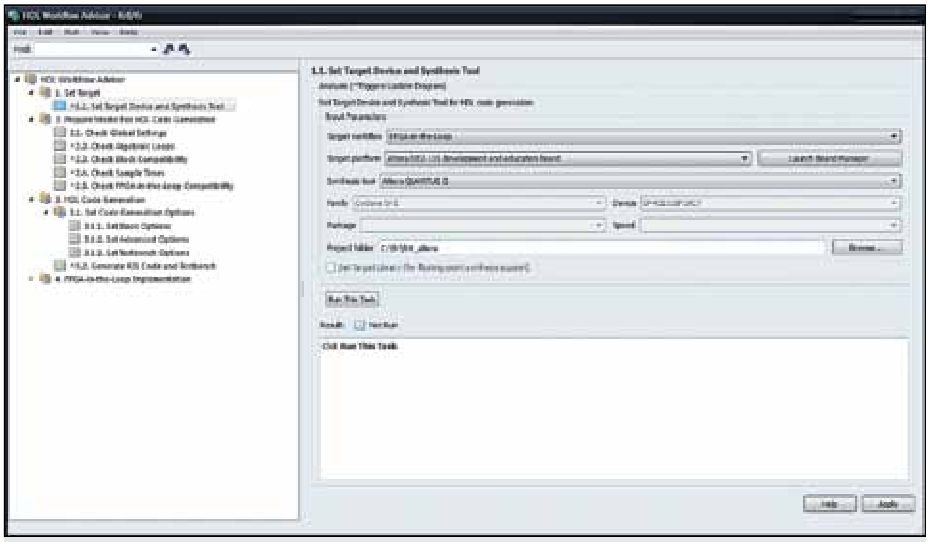

Рис. 12. Программный инструмент HDL Workflow Advisor расширения Simulink

Например, в разделе Set Target выбирается отладочная плата Altera DE2-115 Development and Education Board на ПЛИС серии Cyclone IV EP4CE115. В случае выбора стиля проектирования FPGA-in-the-Loop (замкнутый цикл) отлаживать проект можно непосредственно из Simulink. Проверку поддержки отладочной платы Workflow Advisor можно осуществить на втором этапе подготовки модели для генерации кода Prepare Model For HDL Code Generation в разделе Chek FPGA-in-the-Loop Compatibility. На третьем этапе осуществляется генерация кода.

Выводы

Система MATLAB/Simulink с приложением Simulink HDL Coder может быть эффективно использована для ускорения процесса разработки квантованных КИХ-фильтров в базисе ПЛИС. Если VHDL-код фильтра автоматически извлечен из описания Simulink-модели, то это приводит к использованию встроенных ЦОС-блоков в ПЛИС серии Cyclone II, что обеспечивает разработку высокопроизводительных КИХ-фильтров. ■

Литература

1. Сергиенко А. Б. Цифровая обработка сигналов. СПб.: Питер, 2003.
2. Жуков К. Г. Справочное руководство пользователя Fixed-Point Blockset — http://matlab.exponenta.ru/fixedpoint/book1/1.php
3. Анохин В., Ланнэ А. MATLAB для DSP. Расчет цифровых фильтров с учетом эффектов квантования — http://www.chipinfo.ru/literature/chipnews/200109/1.html
4. Fixed-Point Blockset For Use with Simulink. Modeling Simulation Implementation. User's Guide. Version 2. Fixed-Point Blockset.
5. http://matlab.ru/datasheets/HDL-coder-ru.pdf