Школа схемотехнического проектирования устройств обработки сигналов

Занятие 7.
Реализация вычислительных устройств на ПЛИС.

Память человека есть лист белой бумаги:
иногда напишется хорошо, а иногда дурно.
К. Прутков

В этом занятии мы рассмотрим некоторые примеры реализации цифровых устройств на ПЛИС фирмы Altera с использованием языков описания аппаратуры. Языки описания аппаратуры (Hardware Description Language) являются формальной записью, которая может быть использована на всех этапах разработки цифровых электронных систем. Это возможно вследствие того, что язык легко воспринимается как машиной, так и человеком. Он может использоваться на этапах проектирования, верификации, синтеза и тестирования аппаратуры, так же как и для передачи данных о проекте, модификации и сопровождения.

Владимир Стешенко

steshenk@sm.bmstu.ru

Реализация генератора ПСП на ПЛИС

В качестве простого примера реализации цифрового устройства на ПЛИС рассмотрим пример генератора псевдослучайной последовательности (ПСП). Генератор формирования М-последовательностей был написан в виде параметризованной макрофункции, описывающей устройство, упрощенная структура которого показана на рис. 1. Параметрами макрофункции являются длина характеристического многочлена и число, описывающее начальные состояния триггеров.

Ниже приводится листинг описания этой функции на языке описания аппаратуры AHDL [1].

Функция M_GENERATE:

```
%============================================%
%       Функция M_GENERATE                   %
%       Генерирует М-последовательность, заданную %
%       характеристическим многочленом М     %
%                                            %
%       Входы:                               %
%       M                - вход многочлена;  %
%       GENA             - разрешение генерации (default=GND); %
%       CLK              - тактовый вход;    %
%       LOAD             - вход предустановки (default=GND); %
%       Выходы:                              %
%       OUT              - выход М-последовательности %
%                                            %
%       Параметры:                           %
%       M_WIDTH          - длина полинома;   %
%       M_BEGIN          - начальное состояние %
%                          (default=»100000..00»); %
%                                            %
%       Версия 1.0                           %
%============================================%
INCLUDE «lpm_xor.inc»;
INCLUDE «lpm_constant.inc»;
PARAMETERS
(
        M_WIDTH,
        M_BEGIN = 0
);
SUBDESIGN m_generate
(
        M[M_WIDTH-1..0]         : INPUT = GND;
        CLK                     : INPUT;
        GENA                    : INPUT = GND;
        LOAD                    : INPUT = GND;
        OUT                     : OUTPUT;
)
VARIABLE
        dffs[M_WIDTH-2..0]      : DFFE; - триггеры регистра сдвига
        shift_node[M_WIDTH-2..0] : NODE;
        xor_node[M_WIDTH-2..0]  : NODE;
        shiftin, shiftout       : NODE; - вход и выход регистра
сдвига
        IF (USED(M_BEGIN)) GENERATE
                ac : lpm_constant
                WITH (LPM_WIDTH = M_WIDTH, LPM_CVAL-
UE = M_BEGIN);
        END GENERATE;
    BEGIN
        ASSERT (M_WIDTH>0)
        REPORT «Значение параметра M_WIDTH должно быть больше
нуля»
                SEVERITY ERROR;

    % ---------- общие выводы триггеров ----------%
        dffs[].ena = GENA;
        dffs[].clk = CLK;
    % ---------- асинхронные операции ---------- %
        IF (USED(M_BEGIN)) GENERATE
```

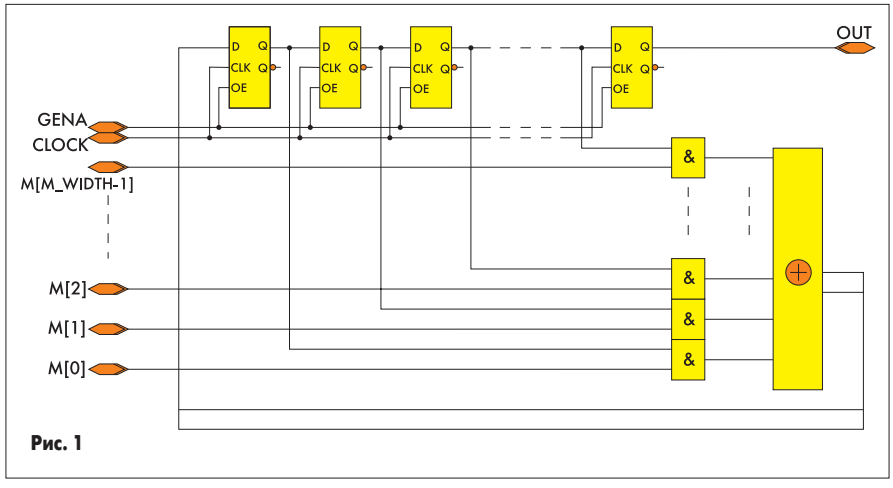

Рис. 1

```
            dffs[].clrn = !load # ac.result[M_WIDTH-2..0];
%           установка начального состояния              %
            dffs[].prn = !load # !ac.result[M_WIDTH-2..0];
%           триггеров, заданное M_BEGIN                 %
            ELSE GENERATE
            dffs[M_WIDTH-3..0].clrn = !load;
%            установка начального состояния             %
            dffs[M_WIDTH-2].prn = !load;
%           триггеров «10000..00                        %
            END GENERATE;

% ---------------------- обратные связи ---------------------- %
            xor_node[] = dffs[] & M[M_WIDTH-2..0];
            shiftin = lpm_xor(xor_node[])
            WITH (LPM_SIZE = M_WIDTH-1, LPM_WIDTH=1);
% ---------------------- операции сдвига ---------------------- %
            shift_node[] = (shiftin, dffs[M_WIDTH-2..1]);
            shiftout = dffs[0];
%--------------------- синхронные операции ----------------- %
            dffs[].d = !load & shift_node[];
%---------------------- подключим выход ------------------- %
            OUT = shiftout;
END;
```

Данная функция имеет следующие входы и выходы:

M — шина (группа выводов) с размером M_WIDTH. На этот вход подаются коэффициенты характеристического многочлена;

CLK — вход тактового сигнала;

GENA — вход разрешения генерации. При GENA= «0» генерация М-последовательности запрещена;

LOAD — вход предустановки. При LOAD= «1» триггеры регистра сдвига устанавливаются в состояние, определяемое параметром M_BEGIN;

OUT — выход М-последовательности.

Для включения данной функции в другие схемы были также созданы включаемый файл, содержащий описание, и графический символ-элемент (см. рис.2).

Файл «**M_GENERATE.INC**»:

```
FUNCTION m_generate (m[(m_width) — (1)..0], clk, gena, load)
WITH (M_WIDTH, M_BEGIN)
                    RETURNS (out);
```

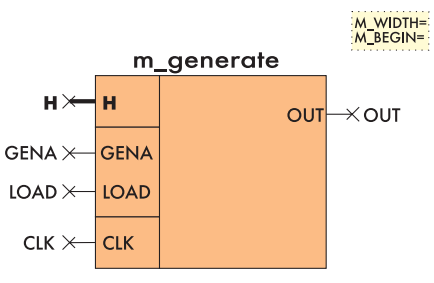

Рис. 2

Для моделирования работы генератора в графическом редакторе пакета MAX+PLUS II [1, 4] была создана тестовая схема, показанная на рис. 3. Результаты моделирования приведены на рис. 4 (на вход одного генератора подан характеристический многочлен M1 = 11001, на вход другого — M2 = 10010000001).

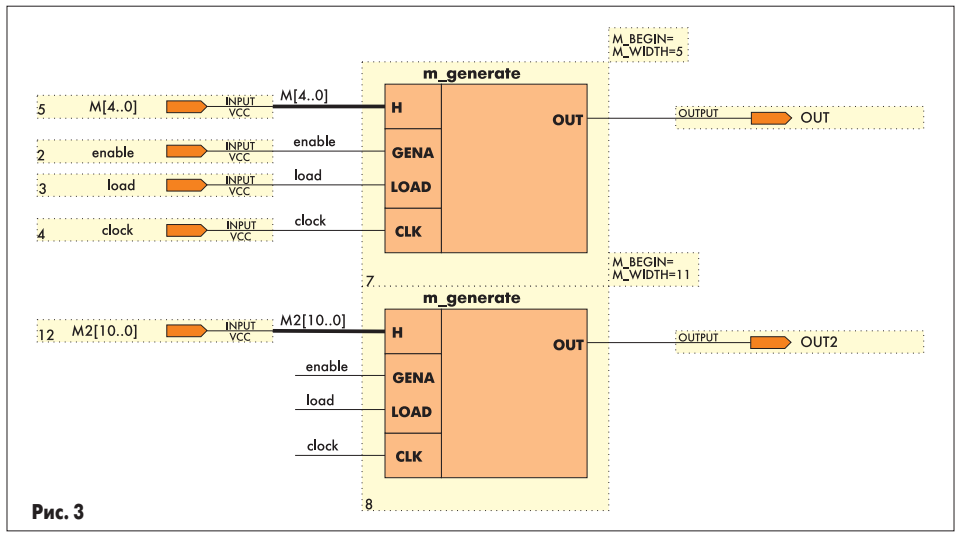

Рис. 3

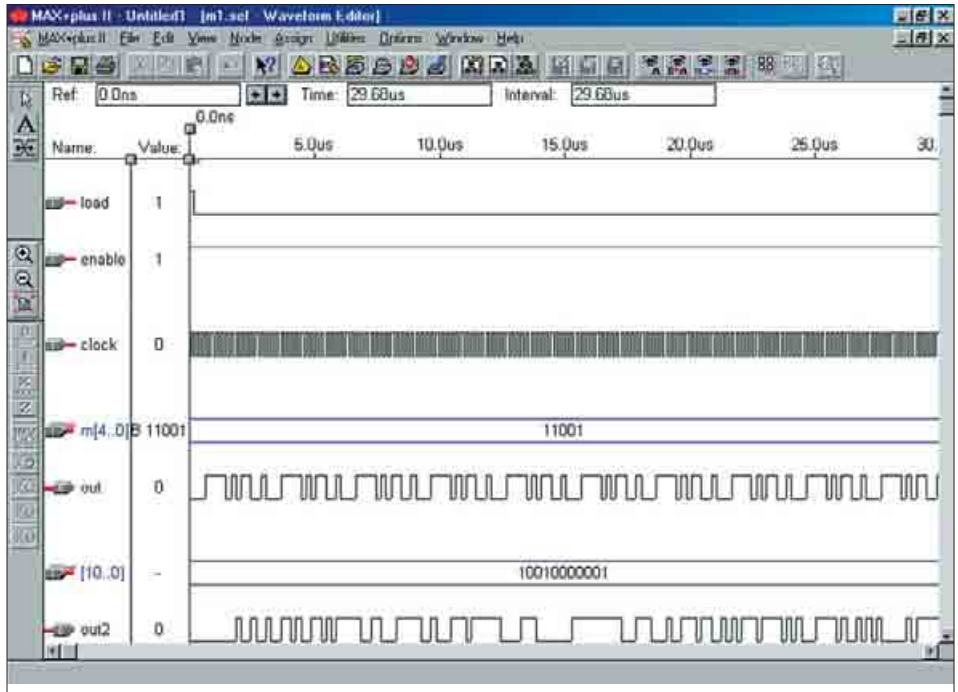

Рис. 4

Реализация цифровых полиноминальных фильтров

Одним из важных классов нелинейных цифровых устройств являются так называемые полиноминальные цифровые фильтры [5, 6]. В последние годы их все чаще применяют в системах обработки изображений.

В общем случае цифровой полиномиальный фильтр размерности *r* и порядка *M*, определяется конечным дискретным рядом Вольтерра (функциональным полиномом) вида

$$y(\mathbf{n}) = h_0 + \sum_{m=1}^{M} y_m(\mathbf{n}) = h_0 + \sum_{m=1}^{M} \sum_{\mathbf{n}_1} \dots \sum_{\mathbf{n}_m} h_m(\mathbf{n}_1, \dots, \mathbf{n}_m) \prod_{i=1}^{m} x(\mathbf{n} - \mathbf{n}_i),$$

где $h_m(n_1, ..., n_m)$ — многомерные импульсные характеристики (ядра) фильтра, зависящие от векторных аргументов $n_i = [n_{i1} ... n_{ir}]$. Фильтры данного вида часто называются также фильтрами Вольтерра.

Непосредственная реализация цифровых полиномиальных фильтров связана с вычислением произведения векторов. Вычислительные затраты могут быть сокращены за счет использования свойства симметрии изотропных фильтров.

Таким образом, путем последовательного применения процедуры декомпозиции полиномиальный фильтр произвольного порядка может быть представлен в виде параллельной структуры, состоящей из линейных фильтров. Операция линейной фильтрации связана с вычислением двухмерной свертки и допускает высокоэффективную реализацию в виде структур систолического типа, матричных и волновых процессоров. Процесс двух-

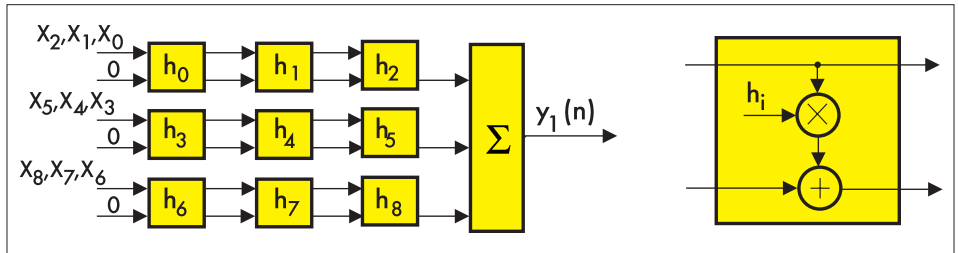

Рис. 5. Систолическая реализация двухмерной линейной свертки

мерной свертки изображения с маской $N \times N$, в свою очередь, можно свести к вычислению N одномерных сверток, что в конечном итоге позволяет выполнять полиномиальную фильтрацию изображений путем параллельного вычисления обычных одномерных сверток. На рис. 5 представлен один из наиболее простых вариантов реализации двухмерной свертки изображения с маской 3 x 3 в виде систолической структуры, состоящей из 9 идентичных процессорных элементов.

Задача двумерной свертки формулируется следующим образом: даны веса $w_{i,j}$ для i,j=1,2...,k, так что k x k — размер ядра, и входное изображение $x_{i,j}$ для i,j=1,2...,n. Требуется вычислить элементы изображения $y_{i,j}$ для i,j=1,2...,n, определяемые в виде

$$y_{i,j} = \sum_{l=1}^{k} \sum_{h=1}^{k} w_{hl} x_{i+h-1, j+l-1}.$$

При k=3 двумерная свертка выполняется в виде трех последовательных одномерных сверток (использующих в качестве весов одну и ту же последовательность (w_{11}, w_{21}, ...,w_{33})):

1. Вычисление (y_{11},y_{12},y_{13},y_{14},...) при использовании (x_{11}, x_{21},x_{31},x_{12},x_{22},x_{32},x_{13},x_{23},x_{33},...) в качестве входной последовательности.
2. Вычисление (y_{21},y_{22},y_{23},y_{24},...) при использовании (x_{21},x_{31},x_{41},x_{22},x_{32},x_{42},x_{23},x_{33},x_{43},...) в качестве входной последовательности.
3. Вычисление (y_{31},y_{32},y_{33},y_{34},...) при использовании (x_{31},x_{41},x_{51},x_{32},x_{42},x_{52},x_{33},x_{43},x_{53},...) в качестве входной последовательности.

Каждую из этих сверток можно выполнить на одномерном систолическом массиве из девяти ячеек. Отметим, что в любом из этих одномерных массивов ячейка занята вычислениями y_{ij} только 1/3 времени.

Для восстановления ячейка с двумя потоками должна быть 8-разрядной для входных данных и иметь 12 бит для представления весов. Для сетки 32 x 32 пикселей в каждом блоке памяти требуется хранить 1024 различных весовых коэффициента и иметь 10-разрядную адресную шину для указания положения каждой точки. 12-разрядный умножитель и 24-разрядный сумматор для получения промежуточных результатов вполне обеспечивают сохранение требуемой точности в процессе вычислений.

Для выполнения многомерной свертки требуется единственная модификация базовой ячейки с двумя потоками — добавление требуемого числа систолических входных потоков и соответствующее увеличение размера мультиплексора для выбора данных.

При решении задач фильтрации изображения с целью удаления шумов, восстановления изображения или улучшения его качества приходится иметь дело с данными, характеризующимися широким динамическим диапазоном. Поэтому в качестве формата данных необходимо использовать числа с плавающей запятой.

Для реализации систолических структур полиноминальных фильтров наиболее пригодны ПЛИС семейства FLEX10K, содержащие встроенные блоки памяти (EAB — embedded array blocks), предназначенные для эффективной реализации функций памяти и сложных арифметических и логических устройств (умножителей, конечных автоматов, цифровых фильтров и т. д.). Один такой блок имеет емкость 2 килобита и позволяет сформировать память с организацией 2048 x 1, 1024 x 2, 512 x 4 или 256 x 8, работающую с циклом 12–14 нс. Использование ВБП значительно повышает эффективность и быстродействие создания сложных логических устройств, например, умножителей. Так, каждый ВБП может выполнять функции умножителя 4 x 4, 5 x 3 или 6 x 2.

На рис. 6 приведена реализация структуры на ПЛИС, а на рис. 7 — результаты моделирования систолической структуры в среде MAX PLUS II.

Примеры описания цифровых схем на VHDL

Наиболее универсальным и распространенным языком описания аппаратуры является VHDL. На этом языке возможно как поведенческое, так структурное и потоковое описание цифровых схем.

Язык VHDL используется во многих системах для моделирования цифровых схем, про-

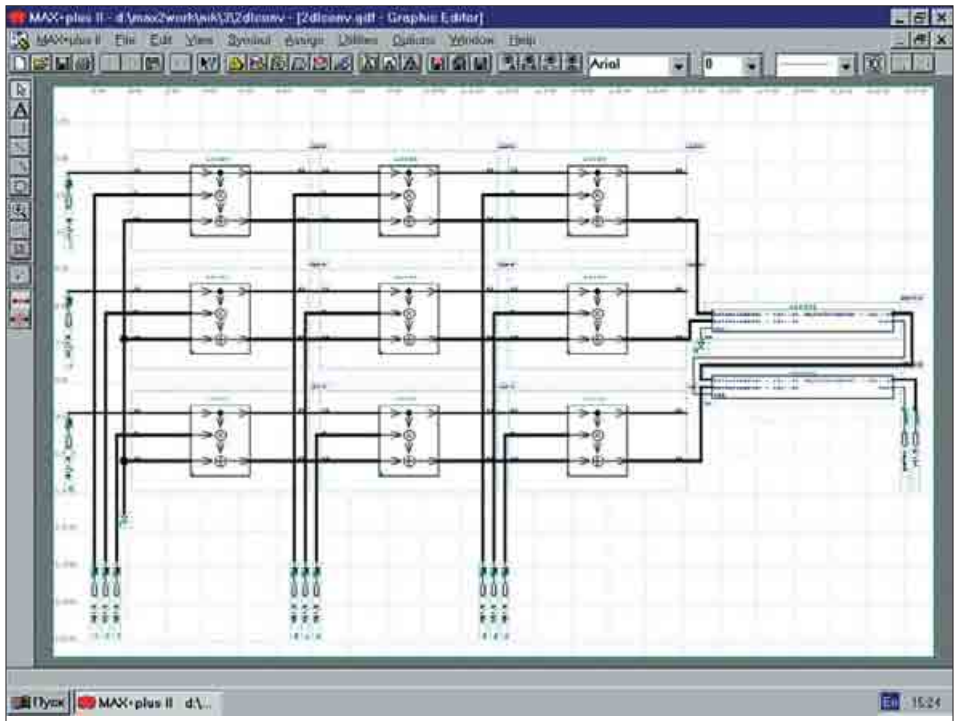

Рис. 6. Реализация систолической структуры на ПЛИС

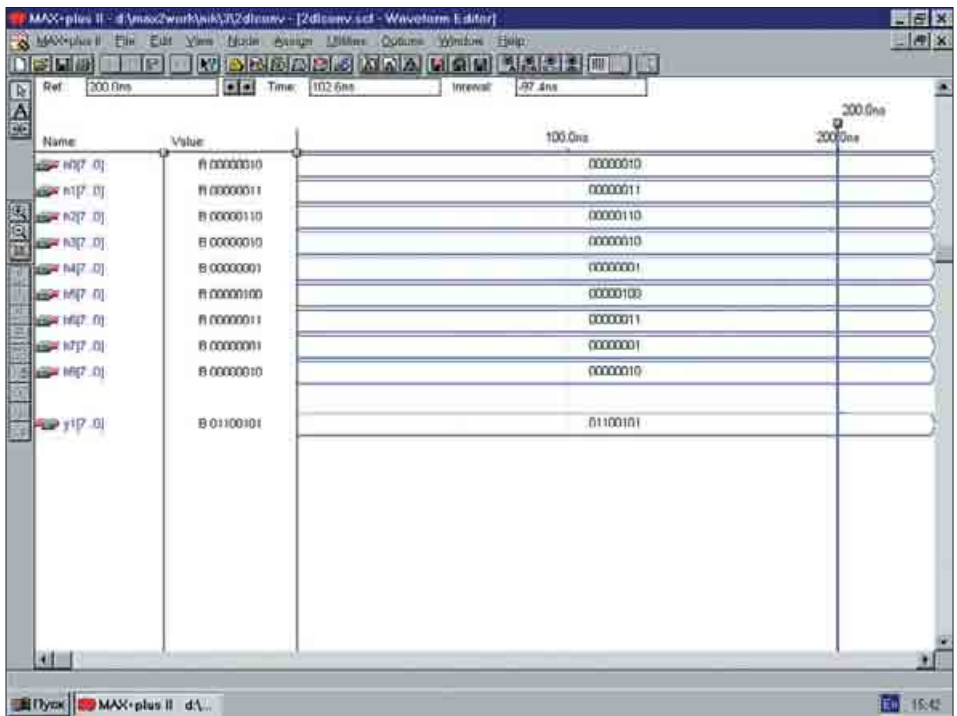

Рис. 7. Результаты моделирования систолической структуры на ПЛИС

ектирования программируемых логических интегральных микросхем, базовых матричных кристаллов, заказных интегральных микросхем.

Рассмотрим некоторые примеры описания цифровых схем на VHDL.

Первый — описание цифрового автомата преобразователя параллельного кода в последовательный. Преобразователь кода представляет собой устройство, на вход которого подается n-битное число в параллельном коде «d», сигнал загрузки «load» и синхроимпульсы «clk». По сигналу загрузки происходит запись входного слова во внутренний регистр и последовательная выдача в течение *n* тактов этого входного слова в последовательном коде на выходе «o» синхроимпульсами «oclk». После окончания преобразования на выходе «e» появляется высокий уровень сигнала в течение одного такта. Такого рода преобразователи кода часто используются для управления синтезаторами частот 1104ПЛ1 и им подобными.

Описание этого устройства на языке VHDL приведено ниже:

```
library ieee;
use ieee.std_logic_1164.all;
entity Serial is
 port (
 clk : in STD_LOGIC;
 load : in STD_LOGIC;
 reset: in STD_LOGIC;
 d : in STD_LOGIC_vector (3 downto 0);
 oclk : out STD_LOGIC;
        o : out STD_LOGIC;
 e : out STD_LOGIC
 );
end;
architecture behavioral of Serial is
 type t1 is range 0 to 4;
        signal s : STD_LOGIC_vector (2 downto 0);
 signal i : t1;
begin
process (clk)
        begin
        if reset = '1' then
                i <= 0;
        else
                if (clk'event and clk='1') then
                        if (i = 0 and load = '1') then
                                s(2 downto 0) <= d(3 downto 1);
                                o <= d(0);
                                i <= 4;
                        end if;
                        if (i > 1) then
                                o <= s(0);
                                s(1 downto 0) <= s(2 downto 1);
                                i <= i — 1;
                        end if;
                        if (i = 1) then
                                e <= '1';
                                i <= 0;
                        else
                                e <= '0';
                        end if;
                end if;
        end if;
        if i>0 then
                oclk <= not clk;
        else
                oclk <= '0';
        end if;
end process;
end behavioral;
```

По переднему фронту синхроимпульса «clk» при высоком уровне на входе загрузки происходит загрузка трех старших бит входного слова d[3..1] во временный регистр s[2..0]. Младший бит входного слова d[0] подается на выход «o». На выходе «oclk» появляются синхроимпульсы. На сигнале «i» собран внутренний счетчик, выдающий сигнал окончания преобразования «e». При поступлении последующих синхроимпульсов происходит выдача на выход «o» остальных бит входного слова, хранящихся в регистре s[2..0].

Моделирование этого устройства было проведено в системе проектирования OrCAD 9.0.

Для тестирования схемы использовался тест:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity test_serial is end test_serial;
architecture testbench of test_serial is
component serial
 port (
 clk : in std_logic;
 load : in std_logic;
 reset : in std_logic;
 d : in std_logic_vector(3 downto 0);
 oclk : out std_logic;
 o : out std_logic;
 e : out std_logic
 );
end component;
signal clk : std_logic;
signal load : std_logic := '0';
signal reset : std_logic;
signal d : std_logic_vector(3 downto 0);
signal oclk : std_logic;
signal o : std_logic;
signal e : std_logic;
begin
        process begin
                for i in 0 to 50 loop
                        clk <= '0'; wait for 5 ns;
                        clk <= '1'; wait for 5 ns;
                end loop;
        end process;
        process begin
                reset <= '1'; wait for 10 ns;
                reset <= '0' ;
                load <= '1';
                d <= «1010»; wait for 10 ns;
                load <= '0';
                d <= «0000»; wait for 500 ns;
        end process;
dut : serial port map (
 clk => clk,
 load => load,
 reset => reset,
 d => d,
 oclk => oclk,
 o => o,
 e => e
 );
end testbench;
```

Результаты моделирования представлены на рис. 8.

В качестве примера описания устройства ЦОС рассмотрим цифровой КИХ-фильтр. Работа цифрового КИХ-фильтра описывается разностным уравнением

$$y_n=A_0x_n + A_1x_{n-1} + A_2x_{n-2} + \ldots,$$

где y_n — реакция системы в момент времени n,

x_n — входное воздействие

A_i — Весовой коэффициент i-й входной переменной.

На VHDL описание фильтра имеет вид:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity f is
port (
                din: in std_logic_vector(7 downto 0);
                sout: out std_logic_vector(15 downto 0);
                r: in std_logic;
                c: in std_logic
                );
end f;
architecture behavior of f is
constant h00 : std_logic_vector(7 downto 0) := «00000000»;
constant h01 : std_logic_vector(7 downto 0) := «00000001»;
constant h02 : std_logic_vector(7 downto 0) := «00000100»;
constant h03 : std_logic_vector(7 downto 0) := «00001111»;
constant h04 : std_logic_vector(7 downto 0) := «00100100»;
constant h05 : std_logic_vector(7 downto 0) := «01000010»;
constant h06 : std_logic_vector(7 downto 0) := «01100100»;
constant h07 : std_logic_vector(7 downto 0) := «01111100»;
constant h08 : std_logic_vector(7 downto 0) := «01111111»;
constant h09 : std_logic_vector(7 downto 0) := «01101010»;
constant h10 : std_logic_vector(7 downto 0) := «01000010»;
constant h11 : std_logic_vector(7 downto 0) := «00010011»;
constant h12 : std_logic_vector(7 downto 0) := «11101100»;
constant h13 : std_logic_vector(7 downto 0) := «11010110»;
constant h14 : std_logic_vector(7 downto 0) := «11010101»;
constant h15 : std_logic_vector(7 downto 0) := «11100011»;
constant h16 : std_logic_vector(7 downto 0) := «11110111»;
constant h17 : std_logic_vector(7 downto 0) := «00001010»;
constant h18 : std_logic_vector(7 downto 0) := «00010100»;
constant h19 : std_logic_vector(7 downto 0) := «00010011»;
constant h20 : std_logic_vector(7 downto 0) := «00001100»;
constant h21 : std_logic_vector(7 downto 0) := «00000010»;
constant h22 : std_logic_vector(7 downto 0) := «11111000»;
constant h23 : std_logic_vector(7 downto 0) := «11110101»;
signal x00, x01, x02, x03, x04, x05, x06, x07,
                x08, x09, x10, x11, x12, x13, x14, x15,
                x16, x17, x18, x19, x20, x21,
                x22, x23 : std_logic_vector(7 downto 0);
```

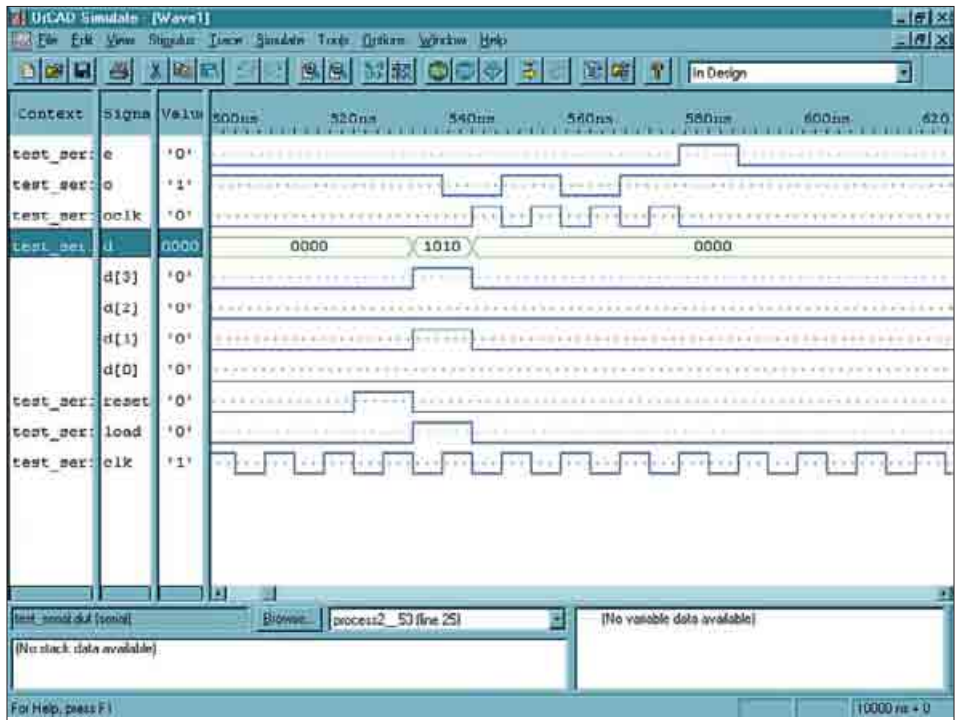

Рис. 8.

```
signal m00, m01, m02, m03, m04, m05, m06, m07,
        m08, m09, m10, m11, m12, m13, m14, m15,
        m16, m17, m18, m19, m20, m21,
        m22, m23 : std_logic_vector(15 downto 0);
begin
m00 <= (signed(x00)*signed(h00));
m01 <= (signed(x01)*signed(h01));
m02 <= (signed(x02)*signed(h02));
m03 <= (signed(x03)*signed(h03));
m04 <= (signed(x04)*signed(h04));
m05 <= (signed(x05)*signed(h05));
m06 <= (signed(x06)*signed(h06));
m07 <= (signed(x07)*signed(h07));
m08 <= (signed(x08)*signed(h08));
m09 <= (signed(x09)*signed(h09));
m10 <= (signed(x10)*signed(h10));
m11 <= (signed(x11)*signed(h11));
m12 <= (signed(x12)*signed(h12));
m13 <= (signed(x13)*signed(h13));
m14 <= (signed(x14)*signed(h14));
m15 <= (signed(x15)*signed(h15));
m16 <= (signed(x16)*signed(h16));
m17 <= (signed(x17)*signed(h17));
m18 <= (signed(x18)*signed(h18));
m19 <= (signed(x19)*signed(h19));
m20 <= (signed(x20)*signed(h20));
m21 <= (signed(x21)*signed(h21));
m22 <= (signed(x22)*signed(h22));
m23 <= (signed(x23)*signed(h23));
sout <= (signed(m00)+signed(m01)+signed(m02)+signed(m03)
        +signed(m04)+signed(m05)+signed(m06)+signed(m07)
        +signed(m08)+signed(m09)+signed(m10)+signed(m11)
        +signed(m12)+signed(m13)+signed(m14)+signed(m15)
        +signed(m16)+signed(m17)+signed(m18)+signed(m19)
        +signed(m20)+signed(m21)+signed(m22)+signed(m23));
process(c,r)
        begin
        if r='1' then
                x00 <= (others => '0');
                x01 <= (others => '0');
                x02 <= (others => '0');
                x03 <= (others => '0');
                x04 <= (others => '0');
                x05 <= (others => '0');
                x06 <= (others => '0');
                x07 <= (others => '0');
                x08 <= (others => '0');
                x09 <= (others => '0');
                x10 <= (others => '0');
                x11 <= (others => '0');
                x12 <= (others => '0');
                x13 <= (others => '0');
                x14 <= (others => '0');
                x15 <= (others => '0');
                x16 <= (others => '0');
                x17 <= (others => '0');
                x18 <= (others => '0');
                x19 <= (others => '0');
                x20 <= (others => '0');
                x21 <= (others => '0');
                x22 <= (others => '0');
                x23 <= (others => '0');
        elsif (c'event and c='1') then
                x00(7 downto 0) <= din(7 downto 0);
                x01(7 downto 0) <= x00(7 downto 0);
                x02(7 downto 0) <= x01(7 downto 0);
                x03(7 downto 0) <= x02(7 downto 0);
                x04(7 downto 0) <= x03(7 downto 0);
                x05(7 downto 0) <= x04(7 downto 0);
                x06(7 downto 0) <= x05(7 downto 0);
                x07(7 downto 0) <= x06(7 downto 0);
                x08(7 downto 0) <= x07(7 downto 0);
                x09(7 downto 0) <= x08(7 downto 0);
                x10(7 downto 0) <= x09(7 downto 0);
                x11(7 downto 0) <= x10(7 downto 0);
                x12(7 downto 0) <= x11(7 downto 0);
                x13(7 downto 0) <= x12(7 downto 0);
                x14(7 downto 0) <= x13(7 downto 0);
                x15(7 downto 0) <= x14(7 downto 0);
                x16(7 downto 0) <= x15(7 downto 0);
                x17(7 downto 0) <= x16(7 downto 0);
                x18(7 downto 0) <= x17(7 downto 0);
                x19(7 downto 0) <= x18(7 downto 0);
                x20(7 downto 0) <= x19(7 downto 0);
                x21(7 downto 0) <= x20(7 downto 0);
                x22(7 downto 0) <= x21(7 downto 0);
                x23(7 downto 0) <= x22(7 downto 0);
        end if;
        end process;
end behavior;
```

Входные данные считываются с входа din[7..0] в дополнительном коде по переднему фронту синхросигнала «c»

На сигналах x0 ÷ x23 построен сдвиговый регистр, обеспечивающий задержку данных на 24 такта. Сигналы с регистров умножаются на весовые коэффициенты h0 ÷ h23 и суммируются.

Для тестирования схемы использован тест:

```
-- Test bench shell
library ieee;
use ieee.std_logic_1164.all;
entity test_f is end test_f;
architecture testbench of test_f is
component f
 port (
 din : in std_logic_vector(7 downto 0);
 sout : out std_logic_vector(15 downto 0);
 r : in std_logic;
 c : in std_logic
 );
end component;
signal din : std_logic_vector(7 downto 0);
signal sout : std_logic_vector(15 downto 0);
signal r : std_logic;
signal c : std_logic;
begin
        process begin
                for i in 0 to 50 loop
                        c <= '0'; wait for 5 ns;
                        c <= '1'; wait for 5 ns;
                end loop;
        end process;
        process begin
                r <= '1'; wait for 10 ns;
                r <= '0' ;
                din <= «00000001»; wait for 10 ns;
                din <= «00000000»; wait for 500 ns;
        end process;
dut : f port map (
 din => din,
 sout => sout,
 r => r,
 c => c
 );
end testbench;
```

Тест моделирует подачу на цифровой фильтр аналога δ-функции. На выходе фильтра — его импульсная характеристика.

Результаты моделирования представлены на рис. 9.

В следующем занятии мы продолжим рассмотрение реализации устройств обработки сигналов на ПЛИС. ■

Литература

1. Стешенко В. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов. М.: Додека, 2000 .
2. Угрюмов Е. П. Цифровая схемотехника. СПб.: БХВ — Санкт-Петербург, 2000.
3. Стешенко В. Б. Школа схемотехнического проектирования устройств обработки сигналов // Компоненты и технологии, № 3–6, 2000 г.
4. Стешенко В. Школа разработки аппаратуры цифровой обработки сигналов на ПЛИС // Chip News, 1999, № 8–10, 2000, № 1, 3–5.
5. Щербаков М. А., Стешенко В. Б., Губанов Д. А. Цифровая полиноминальная фильтрация: алгоритмы и реализация на ПЛИС // Инженерная микроэлектроника, № 1(3), март 1999. С.12–17.
6. Щербаков М. А., Стешенко В. Б., Губанов Д. А. Цифровая полиноминальная фильтрация в реальном масштабе времени: алгоритмы и пути реализации на ПЛИС // Цифровая обработка сигналов, № 1, 2000. С. 19–26.

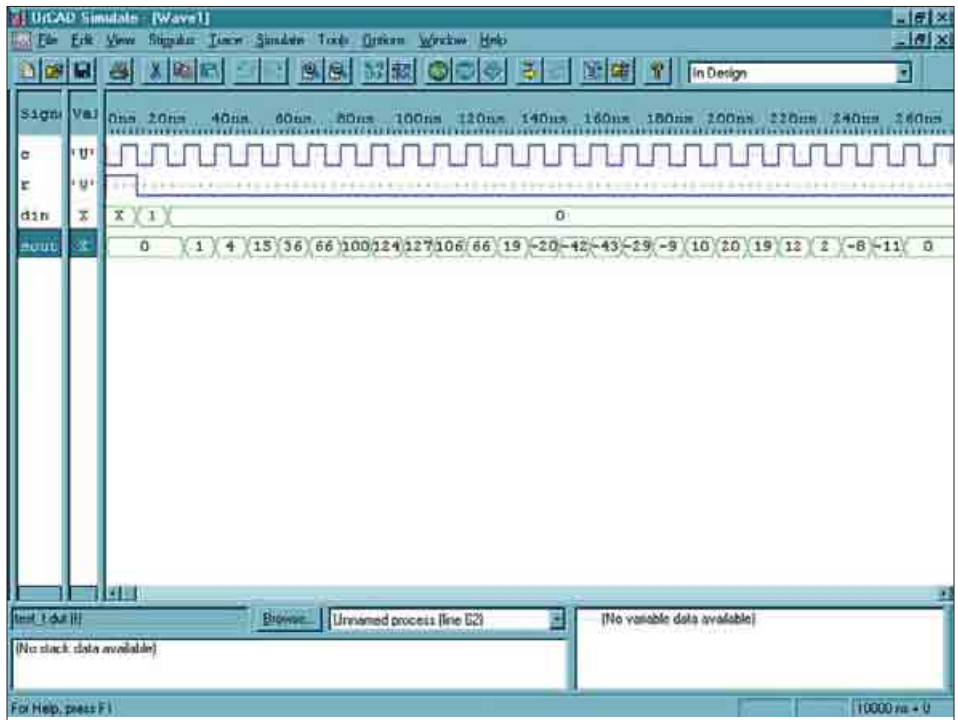

Рис. 9.