# Лабораторная работа 11

## Синтаксические конструкции

### Реализация линии задержки

Напомним, что линия задержки на $N$ отсчетов реализуется на основе регистра сдвига следующим образом (сигнал delay_line должен быть объявлен как массив размера $N$ с элементами необходимой разрядности):

```
process (clk) begin
      if rising_edge(clk) then
            if sample_in_dv='1' then
                  delay_line <= sample_in & delay_line(N-1 downto 1);
            end if;
      end if;
end process;
sample_out <= delay_line(0);
```

Обратим внимание на то, что в приведенном примере отсутствует сигнал сброса.

При реализации линии задержки в ПЛИС интересен результат использования сигнала сброса:

```
process (clk) begin
      if rising_edge(clk) then
            if srst='1' then
                  delay_line <= (others => (others => '0'));
            else
                  if sample_in_dv='1' then
                        delay_line <= sample_in & delay_line(N-1 downto 1);
                  end if;
            end if;
      end if;
end process;
sample_out <= delay_line(0);
```

Казалось бы, совсем незначительное усложнение процесса, приводит к серьезным изменениям результатов разводки. Это связано с тем, что реализация линии задержки по умолчанию реализовывается на логике общего назначения, а точнее на таблицах истинности слайсов типа SLICEM (напомним, что эти слайсы могут работать в режиме ячеек памяти). При этом в FPGA Xilinx у таблиц истинности слайсов физически отсутствует сигнал сброса. Это значит, что его потребуется реализовывать явно, т.е. с использованием дополнительной логики общего назначения. В результате добавление сигнала сброса в синтаксис описания линии задержки приведет к значительному увеличению требуемых ресурсов FPGA.

Заметим также, что линии задержки можно (а в случае больших $N$ – и целесообразно) реализовывать на встроенных блоках памяти FPGA. Для этого удобно использовать IP Core Generator.

## Задание к лабораторной работе 11

Реализовать ФНЧ на основе скользящего среднего.

Необходимо реализовать:

1. *ФНЧ на основе скользящего среднего и простой тестбенч к нему.
   Размер окна фильтра – 8. Разрядность входных данных – 16.

Тестбенч должен подавать на вход фильтра бесконечный 4-разрядный счетчик (счет на каждый такт тактовой частоты). Задержка результата – 1 такт (т.е. для текущего отсчета, среднее его и семи предыдущих отсчетов появляется на следующий такт).
При реализации скользящего среднего усреднение выполнять в последний момент – перед выводом результата из модуля. Усреднение (деление на 8) выполнять отбрасыванием необходимого числа младших разрядов).

Т.к. сигнал сброса не применяется, сигналы аккумулятора (res) и линии задержки при объявлении инициализировать нулями.

Использовать следующий шаблон для объявления интерфейса модуля и архитектуры:

```
--------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.all;

entity lab121 is
    Port ( clk : in STD_LOGIC;
           din : in STD_LOGIC_VECTOR (15 downto 0);
           dout : out STD_LOGIC_VECTOR (15 downto 0));
end lab121;

architecture Behavioral of lab121 is
    type tdelay is array (7 downto 0) of std_logic_vector(15 downto
0);
    signal sdelay    : tdelay:=(others => (others => '0'));
    signal res       : std_logic_vector(18 downto 0):=(others => '0');
begin
...
--------------------------------------
```

2. *Два варианта линии задержки 16-разрядных чисел на 128 тактов: с и без сигнала сброса.
   Реализации должны различаться только наличием сигнала сброса. Файл *.xdc создавать не нужно. После сборки проекта зафиксировать количество ресурсов, потребовавшееся для построения каждого из вариантов. Открыть схемы разведенных проектов и сравнить реализацию. Убедиться, что при реализации линии задержки потребовались дополнительные ресурсы для сброса.
3. **Сложный тестбенч к первому заданию.
   С помощью приведенного ниже исходного кода Matlab сгенерировать тестовый вектор (отсчеты суммарного сигнала) и записать их в текстовый файл в столбец с помощью функции Matlab dlmwrite(). В тестбенче с помощью исходного кода VHDL (приведен ниже) считать из файла отсчеты и подать их на вход реализованного ФНЧ. Выходные отсчеты ФНЧ записать в новый текстовый файл в столбец (пример см. ниже). Записанные отсчеты считать в Matlab с помощью функции dlmread() и сравнить результаты выполнения фильтрации в Matlab и в FPGA. Они должны сойтись.
   Обратите внимание, что в приведенных примерах запись и чтение осуществляются по одному отсчету. Такой подход экономит оперативную память, но сильно замедляет моделирование, за счет частого обращения к жесткому диску. Возможен альтернативный вариант, когда на этапе инициализации считывается в ROM тестбенча весь исходный сигнал, а запись осуществляется сначала в ROM тестбенча, а по окончании моделирования одним обращением к жесткому диску содержимое этого ROM переносится на жесткий диск в файл. Такой подход быстрее, но требует больше оперативной памяти.
   При работе с числами в Matlab необходимо от плавающей точки перейти к псевдоцелым числам (т.е. тип будет по-прежнему double, но знаков после запятой не будет). Для этого после формирования сигнала sig_full умножить его на 1000 и применить к результату функцию floor(). Далее работать с полученным псевдоцелочисленным сигналом.

Модуль чтения из файла

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
LIBRARY std;
USE std.textio.all;

ENTITY ReadFile IS
  generic(
    numOfBits : integer;
    file_name : string := "C:\my_design\my_data_rd.txt"
    );
  port(
    data : out std_logic_vector ((numOfBits-1) downto 0);
    dv: out std_logic;
    rst : in std_logic;
    rfd : in std_logic;
    clk : in std_logic
    );
END ENTITY ReadFile;

--
ARCHITECTURE a OF ReadFile IS
  constant   log_file_rd : string  := file_name;

  file file_rd: TEXT open read_mode is log_file_rd;
BEGIN
  read_data: process(clk,rst)
    variable s: integer;
    variable l : line;
  begin
    if (rst = '1') then
      data <= (others => '0');
      dv <= '0';
    elsif(rising_edge(clk)) then
      if rfd='1' then
        readline(file_rd, l);
        read (l, s);
        data <= CONV_STD_LOGIC_VECTOR(s,numOfBits);
        dv <= '1';
      else
        dv <= '0';
      end if;
    end if;
  end process;
END ARCHITECTURE a;
```

Модуль записи в файл

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
LIBRARY std;
USE std.textio.all;

ENTITY WriteFile_full IS
  generic(
    numOfBits : integer;
    file_name : string := "C:\my_design\my_data_wr.txt"
```

```
    );
  port(
    clk,dv,sign : in std_logic;
    DataIn : in std_logic_vector ((numOfBits-1) downto 0)
    );
END ENTITY WriteFile_full;

--
ARCHITECTURE a OF WriteFile_full IS
  constant   log_file1 : string  := file_name;

  file file_wr: TEXT open write_mode is log_file1;
BEGIN
  write_data : process(clk)
    variable l2 : line;
  begin
    if(rising_edge(clk)) then
      if dv = '1' then
        if sign='0' then
          write (l2, CONV_INTEGER(UNSIGNED(DataIn)));
          writeline(file_wr, l2);
        else
          write (l2, CONV_INTEGER(SIGNED(DataIn)));
          writeline(file_wr, l2);
        end if;
      end if;
    end if;
  end process;
END ARCHITECTURE a;
```

Исходный код Matlab для подзадания 3.

```
clear all
clc
close all
%% Инициализация
fs    = 8000;        % Частота дискретизации
dt    = 1/fs;        % Интервал дискретизации
fmain = 10;          % Частота полезного синуса
ferr  = 1000;        % Частота мешающего синуса
Amain = 10;          % Амплитуда полезного синуса
Aerr  = 1;           % Амплитуда мешающего синуса
n     = (0:8000)';   % Шкала времени (номера отсчетов)
t     = n*dt;        % Шкала времени (в секундах)
ns    = length(n);   % Количество отсчетов сигнала
%% Формирование сигналов
sig_main = Amain*sin(2*pi*fmain*n/fs);  % Полезный синус
sig_err  = Aerr *sin(2*pi*ferr *n/fs);  % Мешающий синус
sig_full = sig_main + sig_err;          % Суммарный сигнал
%% Бегущее среднее
Nwindow = 4;    % Ширина окна
sig_res1 = zeros(ns,1);
for k=Nwindow:ns
    sig_res1(k) = sum(sig_full(k-Nwindow+1:k))/Nwindow;
end
Nwindow = 8;    % Ширина окна
sig_res2 = zeros(ns,1);
for k=Nwindow:ns
```

```
    sig_res2(k) = sum(sig_full(k-Nwindow+1:k))/Nwindow;
end
%% Отображение сигналов
hold all
grid on
plot(t,sig_full,'r')
plot(t,sig_res1,'g')
plot(t,sig_res2,'b')
```

Запись отсчетов исходного сигнала в текстовый файл.

```
%% Запись отсчетов в файл
wr_sig_full = round(sig_full*2048);     % Масштабирование исходного
сигнала для записи в текстовый файл
dlmwrite('data2fpga.dat',wr_sig_full);
```

Чтение результатов моделирования из файла.

```
%% Чтение результатов моделирования алгоритма в FPGA из файла
fpga_result = dlmread('data_from_fpga.dat');
fpga_result = fpga_result/2048;         % Обратное масштабирование
результатов из FPGA для корректного сравнения с результатами Matlab
```

Сравнение результатов обработки сигнала в плавающей точке в Matlab и в фиксированной точке в FPGA.

```
%% Сравнение результатов Matlab с результатами из FPGA
err = fpga_result(Nwindow:ns) - sig_res2(Nwindow:ns);    % Сигнал
ошибки
max(abs(err))    % Максимальная ошибка
mean(abs(err))   % Среднее модуля ошибки
figure(2)        % Эталонный результат в плавающей точке и результат из
FPGA в фиксированной точке
hold all
grid on
plot(fpga_result(Nwindow:ns),'-d')
plot(sig_res2(Nwindow:ns),'--c')
```