Лабораторная работа 3

Синтаксические конструкции

Поведенческое описание устройства. Последовательные операторы

При последовательном подходе к описанию цифрового устройства основной конструкцией является *процесс* (**process**). В теле процесса можно последовательно по времени описывать шаги/этапы разрабатываемого алгоритма.

Синтаксис процесса следующий:

```
[process_label:] process [ (sensitivity_list) ] [is]
      [process_declarations]
      begin
            [sequential statements]
      end process [process_label];
```

Каждый процесс описывается в архитектуре модуля VHDL после ключевого слова **begin**. Важной особенностью процессов является то, что все процессы в модуле выполняются параллельно, но при этом операции внутри процесса выполняются последовательно.

Обычно (но не всегда) у процесса есть список чувствительности: *sensitivity_list*. Список чувствительности содержит список сигналов, при изменении которых будет выполняться процесс. Другими словами, изменение любого сигнала из списка чувствительности приводит к тому, что выполняются все действия внутри процесса. В результате какие-то сигналы обновляют свои значения. В тактируемых процессах в списке чувствительности должна находиться тактовая частота. В нетактируемых процессах в список чувствительности должны входить все сигналы, которыми оперирует данный процесс.

В *process_declarations* выполняется объявление переменных и констант, которые используются в данном процессе.

Важно понимать, что в процессе все операции над переменными выполняются мгновенно, а операции над сигналами – с задержкой. Поэтому результат операций над переменными доступен к использованию в тот же момент времени, в котором эти операции выполнялись. В то же время результат операций над сигналами доступен через некий интервал времени при следующем запуске процесса.

Условный оператор (if-elsif-else)

Условный оператор выполняет ту или иную последовательность действий в зависимости от одного или нескольких условий.

```
if condition then
      sequential statements;
[elsif condition then
      sequential statements; ]
[else
      sequential statements; ]
end if;
```

Условный оператор if-elsif-else должен находиться в теле процесса.

Оператор множественного выбора (case)

Оператор множественного выбора выполняет одну из последовательностей действий в зависимости от значения определенного выражения.

```
case choice_expression is
      when choices =>
            sequential statements;
      when choices =>
            sequential statements;
      [ when others => sequential statements; ]
end case;
```

Так же, как в случае присваивания с множественным выбором, не может существовать два одинаковых условия выбора и должен применяться оператор **when others**, когда в *choices* охвачены не все возможные значения *choice_expression*.

Оператор множественного выбора должен находиться в теле процесса.

Структурное описание устройства. Компоненты

На структурном уровне абстракции описание цифрового устройства выполняется путем указания того, какие модули используются и как они соединены между собой. Для этого в языке VHDL предусмотрены специальные синтаксические конструкции.

В терминологии VHDL устройства, которые соединяются в проекте, называются компонентами. Для структурного описания цифрового устройства необходимо описать подключаемые компоненты, объявить сигналы, с помощью которых компоненты будут соединены, подключить компоненты.

Объявление сигналов и компонентов выполняется в архитектуре модуля VHDL до ключевого слова **begin**, подключение компонентов – после ключевого слова **begin**.

```
architecture architecture_name of NAME_OF_ENTITY is
[Declarations]
Объявление сигналов и компонентов.
begin
[Statements]
Подключение компонентов.
end architecture_name;
```

Синтаксис объявления компонента выглядит почти также как объявление интерфейса модуля с одним отличием: вместо ключевого слова **entity** объявление начинается с ключевого слова **component**:

```
component NAME_OF_COMPONENT [is] [generic (generic_declarations);]
      [port (signal_names: mode type;
             signal_names: mode type;
                  :
            signal_names: mode type);]
end [NAME_OF_COMPONENT];
```

При подключении компонента его портам ставятся в соответствие сигналы или порты модуля, в котором выполняется подключение. Слева от знака установления соответствия указываются порты компонента, справа – сигналы разрабатываемого модуля.

```
instance_label : component name
      port map (
            port_1 => signal_1,
            port_2 => signal_2,
            port_3 => open,
                  :
            port_n => signal_n);
```

В случае, когда нет необходимости подключать какой-либо порт компонента, используется ключевое слово **open**. Для того, чтобы при неподключении входного порта компилятор не выдавал

ошибку, такому порту при объявлении модуля должно быть присвоено инициализирующее значение.

Возможна сокращенная запись подключения компонента по номеру портов:

```
port map (signal_1, signal_2,…, signal_n);
```

При таком синтаксисе к первому порту компонента подключается первый сигнал, ко второму – второй и т.д.). Сокращенный синтаксис при подключении компонента является плохим стилем программирования.

Другим способом подключения компонентов является прямой подключение (direct instantiation). В этом случае объявлять компонент не нужно, а подключать его необходимо с помощью ключевого слова **entity** и указания библиотеки, в которой используется данный компонент.

```
instance_label : entity work.component name
      port map (
            port_1 => signal_1,
            port_2 => signal_2,
            port_3 => open,
            :
            port_n => signal_n);
```

Задание к лабораторной работе 3

Реализовать мультиплексор, демультиплексор и коммутатор в ПЛИС. В качестве входов использовать кнопки и слайдеры на плате, в качестве выходов – светодиодные индикаторы.

Используя процессы необходимо реализовать:

1. Двухадресный мультиплексор с четырьмя входами.
   Цифровое устройство должно иметь два адресных входа (addr, слайдеры), четыре информационных входа (din, кнопки) и один информационный выход (dout, светодиод).

   Использовать следующий шаблон для объявления интерфейса модуля:

```
----------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lab31 is
    Port ( din  : in STD_LOGIC_VECTOR (3 downto 0);
           addr : in STD_LOGIC_VECTOR (1 downto 0);
           dout : out STD_LOGIC);
end lab31;
----------------------------------
```

2. Двухадресный демультиплексор с четырьмя выходами.
   Цифровое устройство должно иметь два адресных входа (слайдеры), один информационный вход (кнопка) и четыре информационных выхода (светодиоды).

   Использовать следующий шаблон для объявления интерфейса модуля:

```
----------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lab32 is
    Port ( din  : in STD_LOGIC;
           addr : in STD_LOGIC_VECTOR  (1 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end lab32;
----------------------------------
```

3. *Коммутатор 4x4 одноразрядных сигналов с использованием компонентов.

Коммутатор 4x4 одноразрядных сигналов с использованием мультиплексора и демультиплексора из заданий 1 и 2. Коммутатор должен коммутировать сигнал с одного из четырех входных портов на один из четырех выходных портов в соответствии с заданными их адресами. Цифровое устройство должно иметь четыре адресных входа (addr1, addr2, слайдеры), четыре информационных входа (din, кнопки) и четыре информационных выхода (светодиоды). В проекте должно быть три vhdl-файла, в которых соответственно описаны: мультиплексор, демультиплексор, топ-модуль проекта (подключение мультиплексора и демультиплексора). Для соединения выхода мультиплексора к входу демультиплексора необходимо объявить в архитектуре новый (внутренний) сигнал.

Использовать следующий шаблон для объявления интерфейса модуля и его архитектуры:

```
--------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lab33 is
    Port ( din : in STD_LOGIC_VECTOR (3 downto 0);
           addr1 : in STD_LOGIC_VECTOR (1 downto 0);
           addr2 : in STD_LOGIC_VECTOR (1 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end lab33;

architecture struct of lab33 is
    signal mux2demux : std_logic;

    component lab31 is
        Port ( din : in STD_LOGIC_VECTOR (3 downto 0);
               addr : in STD_LOGIC_VECTOR (1 downto 0);
               dout : out STD_LOGIC);
    end component lab31;

    component lab32 is
        Port ( din  : in STD_LOGIC;
               addr : in STD_LOGIC_VECTOR  (1 downto 0);
               dout : out STD_LOGIC_VECTOR (3 downto 0));
    end component lab32;
begin
--------------------------------------
```

4. **Двухадресный мультиплексор чисел в прямом двоичном коде на семисегментный индикатор.

   Мультиплексор должен коммутировать одно из четырех четырехразрядных входных чисел в прямом двоичном коде на один (правый на плате) семисегментный индикатор. Цифровое устройство должно иметь четыре четырехразрядных входа (din0, din1, din2, din3 – все слайдеры), двухразрядный адресный вход addr и семь выходов на семисегментный индикатор. Также у модуля должен присутствовать выходной сигнал управления анодом всех четырех семисегментных индикаторов на отладочной плате. Этот сигнал должен быть равен an <= “1110”.

   Использовать следующий шаблон для объявления интерфейса модуля:

```
--------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lab34 is
    Port ( din0 : in STD_LOGIC_VECTOR  (3 downto 0);
           din1 : in STD_LOGIC_VECTOR  (3 downto 0);
           din2 : in STD_LOGIC_VECTOR  (3 downto 0);
           din3 : in STD_LOGIC_VECTOR  (3 downto 0);
           addr : in STD_LOGIC_VECTOR  (1 downto 0);
           an   : out STD_LOGIC_VECTOR (3 downto 0);
           dout : out STD_LOGIC_VECTOR (6 downto 0));
end lab34;
--------------------------------------
```

5. **Тестбенч для модуля из задания 2 (результат выполнения задания не подлежит автопроверке).

Тестбенч должен генерировать последовательность 101010… с периодом 10 нс на информационном входе и последовательно переключать адреса 00, 01, 10, 11. Период прохода всех адресов 100 нс. В процессе моделирования убедиться, что выходной сигнал, адрес которого установлен на входе, совпадает со входным информационным сигналом.

Использовать следующий тестбенч:

```
---------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_bench is
--  Port ( );
end test_bench;

architecture struct of test_bench is
    constant INPUT_PERIOD : time := 10 ns;
    constant ADDR_PERIOD  : time := 100 ns;

    signal din      : std_logic;
    signal addr     : std_logic_vector(1 downto 0);
    signal dout     : std_logic_vector(3 downto 0);

    component lab32 is
    Port ( din  : in STD_LOGIC;
           addr : in STD_LOGIC_VECTOR  (1 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
    end component lab32;
begin

  ---------------------------------------------------------------------------
  -- Generate input
  ---------------------------------------------------------------------------
  input_gen : process
  begin
    din <= '0';
    wait for INPUT_PERIOD;
    loop
      din <= '1';
      wait for INPUT_PERIOD/2;
      din <= '0';
      wait for INPUT_PERIOD/2;
    end loop;
  end process input_gen;
  ---------------------------------------------------------------------------
  -- Generate addr
  ---------------------------------------------------------------------------
  addr_gen : process
  begin
      addr <= "00";
      wait for ADDR_PERIOD;
      loop
        addr <= "00";
        wait for ADDR_PERIOD/4;
        addr <= "01";
        wait for ADDR_PERIOD/4;
        addr <= "10";
        wait for ADDR_PERIOD/4;
        addr <= "11";
        wait for ADDR_PERIOD/4;
      end loop;
  end process addr_gen;
  ---------------------------------------------------------------------------
  -- Instantiate the DUT
  ---------------------------------------------------------------------------
  dut : lab32
    port map (
        -- Inputs
        din  => din,
        addr => addr,
        -- Outputs
```

```
                dout  => dout);

        end struct;
        ---------------------------------------


## Switches
#set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
#set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
#set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
#set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
#set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
#set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
#set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
#set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
#set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
#set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
#set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
#set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
#set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
#set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
#set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
#set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]


## LEDs
#set_property PACKAGE_PIN U16 [get_ports {led[0]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
#set_property PACKAGE_PIN E19 [get_ports {led[1]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
#set_property PACKAGE_PIN U19 [get_ports {led[2]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
#set_property PACKAGE_PIN V19 [get_ports {led[3]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]

##7 segment display
#set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
#set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
#set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
```

```
      #set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
#set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
#set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
#set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
#set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

#set_property PACKAGE_PIN V7 [get_ports dp]

      #set_property IOSTANDARD LVCMOS33 [get_ports dp]

#set_property PACKAGE_PIN U2 [get_ports {an[0]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
#set_property PACKAGE_PIN U4 [get_ports {an[1]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
#set_property PACKAGE_PIN V4 [get_ports {an[2]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
#set_property PACKAGE_PIN W4 [get_ports {an[3]}]
      #set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


##Buttons
#set_property PACKAGE_PIN U18 [get_ports btnC]
      #set_property IOSTANDARD LVCMOS33 [get_ports btnC]
#set_property PACKAGE_PIN T18 [get_ports btnU]
      #set_property IOSTANDARD LVCMOS33 [get_ports btnU]
#set_property PACKAGE_PIN W19 [get_ports btnL]
      #set_property IOSTANDARD LVCMOS33 [get_ports btnL]
#set_property PACKAGE_PIN T17 [get_ports btnR]
      #set_property IOSTANDARD LVCMOS33 [get_ports btnR]
```