Лабораторная работа 2

Синтаксические конструкции

Типы данных

В библиотеках VHDL определено большое количество различных типов данных. Наиболее популярными являются типы **std_logic** и **std_logic_vector**. Данные типа **std_logic** одноразрядные, могут принимать следующие значения: 0, 1, X (неизвестно), U (неинициализировано), - (не важно), Z (высокий импеданс), L (слабый 0), H (слабая 1), W (слабое неизвестное). Данные типа **std_logic_vector** многоразрядные, представляют из себя набор из нескольких **std_logic** (примеры объявления данных см. ниже).

Константы

Константы имеют заданное значение определенного типа данных и не могут менять свое значение в процессе моделирования или работы устройства.

```
constant const_name_1, const_name_2,… : type := value;
```

Объявление одной или нескольких констант одного типа с одинаковыми значениями выполняется в архитектуре модуля до ключевого слова **begin** и начинается с ключевого слова **constant**. Далее следует перечисление названий констант, их типа данных и значения.

Переменные

Объявление переменных выполняется также, как и констант, при этом необязательно указывать ее начальное значение. Переменные можно объявлять только в процессе (см. ниже). Для присвоения значений переменным используется оператор :=. При выполнении операций над переменными, обновление значения переменной выполняется БЕЗ задержки, мгновенно.

```
variable var_name_1, var_name_2,… : type [:= value];
```

Сигналы

Сигналы объявляются также и там же, где и константы, начальное значение указывать необязательно. Для присвоения значений сигналам используется оператор <=. При выполнении операций над сигналами, обновление значения сигнала выполняется с некоторой задержкой, а не мгновенно, как в случае переменных.

```
signal sig_name_1, sig_name_2,… : type [:= value];

signal data_valid: std_logic;

signal data: std_logic_vector (31 downto 0):= x"deadbeaf";
```

При разработке синтезируемых цифровых устройств полезно использовать сигналы, а не переменные, т.к. снижается вероятность ошибки при реализации.

Указание системы счисления

При инициализации сигнала (переменной, константы) или при присвоении ему какого-либо значения можно использовать двоичную, восьмеричную и шестнадцатеричную системы счисления (СС). При этом один символ двоичной СС соответствует одному разряду сигнала, один символ восьмеричной СС – трем разрядам сигнала, один символ шестнадцатеричной СС – четырем разрядам сигнала. Это значит, что, например, в VHDL 16-разрядное число нельзя записать в восьмеричной СС.

Ниже приведены примеры присваивания чисел в различных СС. Подразумевается, что все сигналы объявлены как std_logic_vector необходимой разрядности. Необходимо обратить

внимание на то, что все числа заключены в двойные кавычки (это справедливо даже для случая, когда сигнал имеет только один разряд, но объявлен как std_logic_vector).

```
sig_bin <= "010101";    -- 21
sig_oct <= o"025";      -- 21
sig_hex <= x"deadbeaf"; -- 3735928495
```

В случае, когда сигнал объявлен как std_logic (сигнал одноразрядный), используется двоичная СС и одинарные кавычки:

```
sig_std_logic <= '1';
```

Уровни представления (абстракции)

Цифровое устройство может быть представлено (описано) на различных уровнях абстракции. Уровни абстракции различаются степенью детализации представления (отдельный транзистор, регистр, АЛУ и контроллеры, микропроцессоры) и акцентами (как соединены компоненты устройства, какие функции выполняет устройство, как происходит распространение сигнала по чипу). Различные способы описания необходимы для упрощения разработки, отладки и описания сложных цифровых устройств.

Различают следующие уровни абстракции: поведенческий и структурный.

Самым высоким уровнем абстракции является поведенческий. На этом уровне цифровое устройство описывается с позиций тех функций, которые оно реализует. При этом не интересуются ни составом устройства, ни используемой элементной базой и примитивами. Существует два вида поведенческого описания цифрового устройства: алгоритмический и поток данных (data flow). В последнем описание устройства обычно выполняется путем указания того, каким образом данные одного регистра передаются на другие регистры (отсюда другое название способа описания: RTL – register transfer level). На RTL уровне используются различные параллельные присваивания.[1] При алгоритмическом подходе последовательно описываются необходимые шаги реализуемого алгоритма и пользуются последовательными присваиваниями сигналов и процессами.

На структурном уровне абстракции описание цифрового устройства выполняется путем указания того, какие модули используются и как они соединены между собой. Обычно структурное описание ближе к конкретной реализации цифрового устройства в заданном чипе.[2]

Язык VHDL (как и Verilog) поддерживает описание цифровых устройств на всех описанных уровнях абстракции, включая их смесь.

Поведенческое описания устройства. Параллельные операторы

Параллельное присваивание (<=)

Синтаксис простого параллельного присваивания следующий:

```
target_signal <= expression;
```

Здесь выражение справа от оператора присваивания, присваивается необходимому сигналу. Операция присваивания выполняется только тогда, когда меняется любой из сигналов, входящих в *expression*.

Условное присваивание (when-else)

Синтаксис условного параллельного присваивания следующий:

---

[1] В лабораторной работе №1 использовалось поведенческое описание (RTL).

[2] Структурное описание цифровых устройств будет рассмотрено в лабораторной работе №3.

```
target_signal <=  expression when condition else
                  expression when condition else
                        :
                  expression;
```

Возможны следующие операторы сравнения:

Оператор	Описание оператора
=	равно
/=	не равно
<	меньше, чем
>	больше, чем
<=	не больше, чем
>=	не меньше, чем

Сигнал, в который выполняется присваивание, примет то значение, для которого первым выполнится условие сравнения. В случае, когда ни одно из условий не выполняется, будет присвоено последнее выражение.

Условное параллельное присваивание будет выполняться только тогда, когда меняются хотя бы один из сигналов справа от оператора <=.

Присваивание с множественным выбором (with-select-when)

Синтаксис присваивания с множественным выбором следующий:

```
with choice_expression select target_signal <=
      expression0 when choices,
      expression1 when choices,
      :
      expressionN when [choices] [others];
```

Такой тип присваивания удобен для табличной реализации логических функций.

В зависимости от значения `choice_expression` сигналу `target_signal` будет присвоено соответствующее значение. Не может существовать два одинаковых условия выбора. Также значения `choices` должны охватывать весь диапазон возможных значений `choice_expression` или в последнем выборе должен присутствовать оператор **`others`** (более частая ситуация).

Задание к лабораторной работе 2

Реализовать шифраторы, дешифраторы, кодопреобразователи и фунцию $\log_2(x)$ в ПЛИС. В качестве входов использовать кнопки и слайдеры на плате, в качестве выходов – светодиодные индикаторы.

Также необходимо реализовать корректную привязку ножек устройства к ножкам ПЛИС (см. *“Исходные данные для конфигурирования ПЛИС.pdf”*)

Необходимо реализовать:

1. Трехвходовый дешифратор.
   Цифровое устройство должно иметь три входа (слайдеры, din) и восемь выходов (светодиоды, dout).

   Использовать следующий шаблон для объявления интерфейса модуля:
   ```
   ----------------------------------------
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   ```

```
entity lab21 is
    Port ( din  : in STD_LOGIC_VECTOR  (2 downto 0);
           dout : out STD_LOGIC_VECTOR (7 downto 0));
end lab21;
--------------------------------------
```

2. Восьмивходовый шифратор.

Цифровое устройство должно иметь восемь входов (слайдеры) и четыре выхода (светодиоды). Три младших (dout(2 downto 0)) выхода использовать для случаев корректных данных. При этом должно быть dout(3)=0. В случае подачи некорректных входных данных (запрещенная входная комбинация) должно быть dout <= “1000”.

Использовать следующий шаблон для объявления интерфейса модуля:

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

entity lab22 is
    Port ( din  : in STD_LOGIC_VECTOR  (7 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end lab22;--------------------------------------
```

3. *Кодопреобразователь четырехразрядного прямого двоичного кода в код Грея.

Прямой двоичный код должен задаваться с помощью слайдеров (din), а результат – выводиться на четыре светодиода (dout).

Использовать следующий шаблон для объявления интерфейса модуля:

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

entity lab23 is
    Port ( din  : in STD_LOGIC_VECTOR  (3 downto 0);
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end lab23;
--------------------------------------
```

4. *Кодопреобразователь четырехразрядного прямого двоичного кода в семисегментный код.

Прямой двоичный код должен задаваться с помощью четырех слайдеров (din). Результат преобразования необходимо выводить на один (правый на плате) семисегментный индикатор (dout). Также у модуля должен присутствовать выходной сигнал управления анодом всех четырех семисегментных индикаторов на отладочной плате. Этот сигнал должен быть равен an <= “1110”.

На отладочной плате Basys3 сегменты каждого индикатора (каждой цифры) подключены по схеме с общим анодом, а управление катодами – раздельное. При этом катоды одинаковых сегментов разных индикаторов соединены. Т.е. сигнал на аноде является как бы сигналом выбора данной цифры, тогда как на катоды всех индикаторов поступает одинаковый сигнал.
Для зажигания сегмента на анод необходимо подать 1, а на катод – 0. Однако, на выходах FPGA, которые подключены к анодам, стоят инвертирующие буферы. Таким образом, для того, чтобы на плате Basys 3 зажечь сегмент на анод и на катод необходимо подать 0 (см. ниже пример зажигания цифры ‘0’).
Десятичную точку подключать не надо.

Использовать следующий шаблон для объявления интерфейса модуля и его архитектуры:

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

entity lab24 is
    Port ( din  : in STD_LOGIC_VECTOR (3 downto 0);
           dout : out STD_LOGIC_VECTOR (6 downto 0);
           an   : out STD_LOGIC_VECTOR (3 downto 0));
end lab24;

architecture Behavioral of lab25 is
```

```
begin
    an   <= "1110";

    dout <= not "0111111" when din=X"0" else -- 0
     …
     …
     …

end Behavioral;
---------------------------------------
```

Использовать следующий шаблон xdc-файла для подключения семисегментного индикатора:

Подключение слайдеров, светодиодов и семисегментного индикатора в XDC файле на отладочной плате Basys 3. Раскомментировать необходимые строки.

```
## 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]}]

## 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]}]
#set_property PACKAGE_PIN W18 [get_ports {led[4]}]

 #set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
#set_property PACKAGE_PIN U15 [get_ports {led[5]}]
```

```
 #set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
#set_property PACKAGE_PIN U14 [get_ports {led[6]}]

 #set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
#set_property PACKAGE_PIN V14 [get_ports {led[7]}]

 #set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]

##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]}]
```

5. ** Функцию $\log_2(x)$ для пятиразрядных входов.

   Реализовать функцию логарифма по основанию 2 для 5-разрядных неотрицательных целых чисел на основе приоритетного шифратора. В качестве входов использовать слайдеры, результат выдавать на правую цифру семисегментного индикатора.

   Использовать следующий шаблон для объявления интерфейса модуля:

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

entity lab25 is
    Port ( din  : in STD_LOGIC_VECTOR (4 downto 0);
           dout : out STD_LOGIC_VECTOR (6 downto 0);
           an   : out STD_LOGIC_VECTOR (3 downto 0));
end lab25;
----------------------------------------
```