Лабораторная работа 7

Синтаксические конструкции

Составные типы данных Массивы

Данные составного типа представляют из себя объединение данных другого типа. В VHDL существует два типа составных данных: массивы и записи. Массивы объединяют в себе однотипные данные, а записи – данные разных типов. Для того, чтобы пользоваться составным типом данных, его необходимо сначала объявить.

Объявление типа данных *массив* выглядит следующим образом:

```
type array_name   is array (indexing)     of type_of_element;
```

Например,

```
type MY_INT       is array (15 downto 0)  of std_logic;
type MY_INTS      is array (0 to 5)       of integer;
type MY_RAM       is array (0 to 127)     of std_logic_vector(15 downto 0);

type MY_MATRIX    is array (0 to 4, 0 to 15) of natural; -- Двумерный массив
```

Существует также возможность создавать массивы, не указывая их размера. В этом случае размер массива будет конкретизироваться непосредственно при объявлении сигнала заданного типа.

Объявление типа данных *запись* выглядит следующим образом:

```
type record_name is
record
      element_name : type_of_element;
            :
      element_name : type_of_element;
end record;
```

Например,

```
type MY_RECORD is
record
      ID       : integer range 0 to 99;
      DATA     : std_logic_vector(15 downto 0);
      SIM_TIME : time;
end record;
```

После объявления типа данных можно объявлять сигналы этого типа также, как и сигналы стандартных типов.

```
signal      SIG_MY_INT  : MY_INT;
constant    SIG_MY_INTS : MY_INTS := (0,1,2,3,4);
signal      SIG_MY_RAM  : MY_RAM  := (others => X"7fff");

signal SIG_MY_RECORD : MY_RECORD;
```

Обращение к элементам массива выполняется с помощью круглых скобок, в которых указывается требуемый индекс или диапазон индексов.

```
SIG_MY_RAM(4)
SIG_MY_RAM(45 to 90)
```

Обращение к элементам записи выполняется с помощью точки:

```
SIG_MY_RECORD.DATA
```

Память

Модули памяти в проекте можно создавать как с помощью IP Core Generator, так и непосредственно на VHDL. В свою очередь, при использовании VHDL можно применять стандартные примитивы, макросы или писать “узнаваемый” синтезатором код. Рассмотрим последний вариант применительно к памяти типа RAM.

Сначала необходимо объявить тип данных массив и саму память:

```
type MY_RAM       is array (0 to 127)     of std_logic_vector(15 downto 0);
signal SIG_MY_RAM : MY_RAM;

signal address    : std_logic_vector(6 downto 0);
```

Синтаксис блока памяти типа *“read first”* выглядит следующим образом (сигнал `ram_enable` можно не использовать, т.е. память всегда включена):

```
process (clk)
begin
   if rising_edge(clk) then
      if (ram_enable = '1') then
         if (write_enable = '1') then
            my_ram(conv_integer(unsigned(address))) <= input_data; -- Запись
         end if;
         ram_output <= my_ram (conv_integer(unsigned(address)));   -- Чтение.
Выводим ранее записанное слово
      end if;
   end if;
end process;
```

Синтаксис блока памяти типа *“no change”* выглядит следующим образом (сигнал `ram_enable` можно не использовать, т.е. память всегда включена):

```
process (clk)
begin
   if rising_edge(clk) then
      if (ram_enable = '1') then
         if (write_enable = '1') then
            my_ram(conv_integer(address)) <= input_data; -- Запись
         else
            ram_output <= my_ram(conv_integer(unsigned(address))); -- Чтение
         end if;
      end if;
   end if;
end process;
```

Синтаксис блока памяти типа *“write first”* выглядит следующим образом (сигнал `ram_enable` можно не использовать, т.е. память всегда включена):

```
process (clk)
begin
```

```
   if rising_edge(clk) then
      if (ram_enable = '1') then
         if (write_enable = '1') then
            my_ram (conv_integer(address)) <= input_data;  -- Запись
            ram_output <= input_data; -- Сразу выводим записываемое слово
         else
            ram_output <= my_ram(conv_integer(unsigned(address))); -- Чтение
         end if;
      end if;
   end if;
end process;
```

Применение памяти типа ROM отличается от рассмотренных выше тем, что непосредственно в процессах, описывающих память, присутствует только чтение из памяти, а при объявлении памяти сразу выполняется ее инициализация требуемыми данными.

Отметим, что для повышения быстродействия модуля памяти (особенно, если он состоит из большого количество блоков памяти) на выходе необходимо ставить дополнительные регистры. В FPGA Xilinx блоки памяти имеют такие встроенные регистры, поэтому их использование не увеличивает количество требуемых для реализации логических ресурсов общего назначения. Естественно, задержка вывода данных при использовании выходных регистров памяти увеличивается на один такт.

Параметры модуля. Generic

При объявлении интерфейса модуля VHDL можно указывать не только порты разных типов, но и параметры компиляции модуля. Параметры используются для передачи в модуль определенной (часто, настроечной) информации (например, указание разрядностей портов и внутренних сигналов). Важно отметить, что значения параметров не меняются в процессе работы модуля, значения параметров указываются при подключении компонента. Параметры удобно использовать для создания универсальных модулей.

Объявление параметров компиляции начинается с ключевого слова **generic**:

```
entity name_of_entity is
      [generic (generic_declarations);]
      [port (signal_names: mode type;
             signal_names: mode type;
                   :
            signal_names: mode type);]
end [name_of_entity];
```

При объявлении параметров можно указать их значения по умолчанию. Эти значения будут использованы, если при подключении компонента не будут указаны значения параметров. Например:

```
entity my_entity is
generic(
      word_in_width  : integer := 5;
      word_out_width : integer := 27);
port(
      data_in  : in  std_logic_vector(word_in_width-1  downto 0);
      data_out : out std_logic_vector(word_out_width-1 downto 0));
end my_entity;
```

При подключении компонента, у которого есть параметры, можно (но необязательно, если у параметров есть значения по умолчанию) указать их значения с помощью **generic map** перед **port map**:

```
my_inst : my_entity
generic map(
      word_in_width  => 10,
      word_out_width => 40)
port map(
      data_in  => sig1,
      data_out => sig2);
```

Введение в TCL

Язык Tcl – tool command language, Tcl – это интерпретируемй язык программирования общего назначения. Он применяется для быстрого прототипирования программного обеспечения, разработки GUI, написания скриптов и тестирования. Язык Tcl содержит базовый набор команд, но различные среды разработки (и Vivado тоже) имеют свои собственные надстройки языка для решения своих специализированных задач.

В языке Tcl все трактуется как команда (отсюда и название “command language”): все операции являются командами. Каждая команда может принимать переменное количество параметров, которые могут бысть строками или результатами выполнения других команд.

Язык Tcl предоставляет мощный инструмент для управления, анализа и отладки проектами Vivado. Также этот язык очень удобен для автоматизации процессов создания проекта, моделирования, запуска синтеза и разводки и анализа их результатов.

В общем случае возможны два подхода применения Tcl в Vivado: из консоли Tcl и с использованием скриптов. В этой работе мы рассмотрим первый подход, второй описан в 12-й работе.

Общий синтаксис команды Tcl следующий:

```
CommandName arg0 arg1 …
```

Ниже приведены несколько примеров команд Tcl. Символ # обозначает комментарии.

```
# Отправить текст в стандартный выход (std output)
puts “Start…”
# Создать переменную x и присвоить ей значение 1
set  x 1
# Создать переменную ProjectName и присвоить ей строку
set  ProjectName “MyVivadoProject”
# Вывести содержимое переменной ProjectName
puts $ProjectName
# Использовать результат выполнения одной команды в качестве переменной для другой
set result [expr {$x + $x}]
# Получить путь к текущей папке и присвоить его в переменную
Set ProjectPath [pwd]
```

Проект в среде Vivado организован в виде базы данных, при этом язык Tcl предоставляет возможность задавать этой базе данных различные вопросы. Расширение языка Tcl в среде Vivado описано в документе “Vivado Design Suite Tcl Command Reference Guide”.

Для выполнения команд Tcl необходимо переключиться в консоль Tcl (Рис. 1).

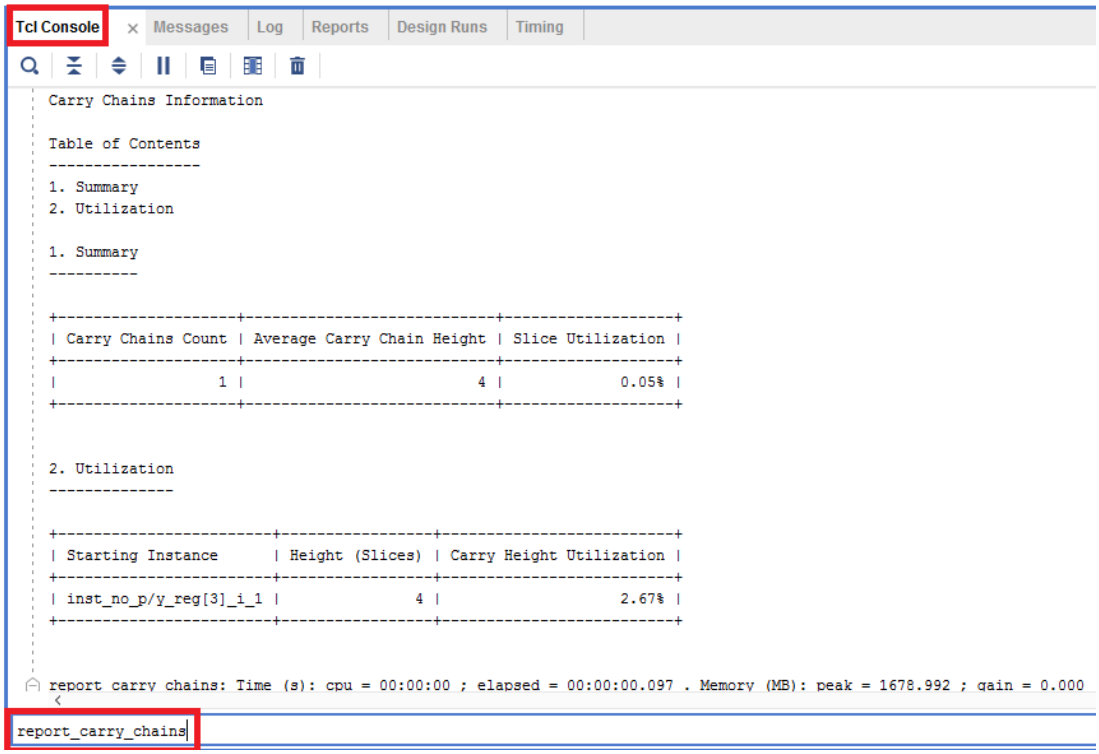

Рис. 1. Консоль Tcl в среде Vivado и команда “report_carry_chains”

Задание к лабораторной работе 7

Реализовать различные типы памяти. В качестве входов использовать кнопки и/или слайдеры, в качестве выходов – светодиоды. Одним из входов разработанного устройства должен быть сигнал тактирования, корректно описанный в XDC файле.

Необходимо реализовать (тип памяти “read first”):

1. Генератор синусоидального колебания на основе ROM (16 слов по 8 битов).
   Устройство должно содержать входы тактирования и сброса и 8-разрядный выход для отсчетов синусоидального сигнала. В устройстве на адресный вход ROM необходимо подавать бесконечный суммирующий на 1 счетчик.
   Память ROM должна содержать 16 отсчетов одного периода синусоидального колебания с частотой, равной 1/16 от частоты дискретизации.
   Колебание в непрерывном виде:

$$s(t) = \sin(2\pi ft).$$

   Отсчеты сигнала с частотой дискретизации F_s и интервалом дискретизации $\Delta t = 1/F_s$:

$$s_n = \sin(2\pi fn\Delta t) = \sin(2\pi fn/F_s) = |f = F_s/16| = \sin(2\pi n/16),\ n = 0, 1, 2, \ldots$$

   Первые 16 отсчетов этого колебания составляют его период и приведены на рис. 1.

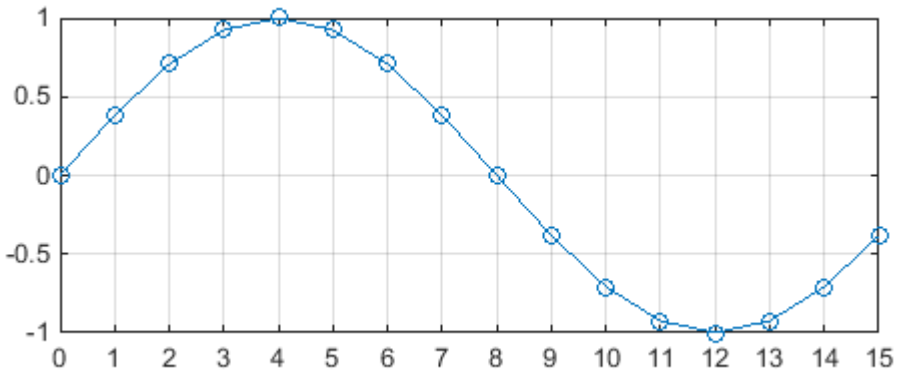

Рис. 1. Отсчеты одного периода синусоидального колебания с частотой $F_s/16$

Значения отсчетов одного периода синусоидального колебания с частотой $F_s/16$ приведены в таблице 1. Для того, чтобы получить двоичное представление значений отсчетов, исходные значения умножены на 127, округлены отбрасыванием знаков после запятой и переведены в двоичный 8-разрядный вид. В разрабатываемом ROM должны храниться значения из последнего столбца таблицы 1.

Отметим, что табличное генерирование синусоидальных колебаний очень часто используется при разработке цифровых смесителей частоты (таких устройств, как DDS (direct digital synthesis), DDC/DUC (digital down/up converter), NCO (numeric controlled oscillator)).

Отметим также, что т.к. не указано требование использовать сигнал разрешения считывания, то это значит, что ROM выдает новый отсчет на каждом такте. Т.е. частота дискретизации равна частоте тактирования и генерируется колебание с частотой 1/16 от тактовой: 100МГц/16. Задержка считывания из ROM должна быть равна 1 такту.

Таблица 1. Значения отсчетов одного периода синусоидального колебания с частотой $F_s/16$

n	≈sin(2πn/16)	≈floor(127*sin(2πn/16))	Двоичное представление отсчетов
0	0	0	0000 0000
1	0,3826	48	0011 0000
2	0,7071	89	0101 1001
3	0,9238	117	0111 0101
4	1	127	0111 1111
5	0,9238	117	0111 0101
6	0,7071	89	0101 1001
7	0,3826	48	0011 0000
8	0	0	0000 0000
9	-0,3826	-49	1100 1111
10	-0,7071	-90	1010 0110
11	-0,9238	-118	1000 1010
12	-1	-127	1000 0001
13	-0,9238	-118	1000 1010
14	-0,7071	-90	1010 0110
15	-0,3826	-49	1100 1111

Использовать следующий шаблон для объявления интерфейса модуля:

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

entity lab71 is
    Port ( clk : in STD_LOGIC;
           srst : in STD_LOGIC;
           dout : out STD_LOGIC_VECTOR (7 downto 0));
end lab71;
-------------------------------------
```

2. RAM 4 слова по 3 бита.

Цифровое устройство должно иметь следующие входы: 2-х разрядный адрес, 3-х разрядный информационный порт, кнопка включения режима записи (we). Устройство должно иметь один 3-

x разрядный информационный выход. На три разряда выходных порта (светодиоды) необходимо выводить содержимое RAM.

Использовать следующий шаблон для объявления интерфейса модуля и его архитектуры:

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

entity lab72 is
    Port ( clk : in STD_LOGIC;
           we : in STD_LOGIC;
           address : in STD_LOGIC_VECTOR (1 downto 0);
           din : in STD_LOGIC_VECTOR (2 downto 0);
           dout : out STD_LOGIC_VECTOR (2 downto 0));
end lab72;

architecture a of lab72 is
    type MY_RAM is array (0 to 3) of std_logic_vector(2 downto 0);
    signal SIG_MY_RAM : MY_RAM:=(others => (others=> '0'));
...
-------------------------------------
```

Обратите внимание на инициализацию всеми нулями памяти. Эта операция не является строго обязательной и может в небольшой степени занимать дополнительные ресурсы FPGA. В текущем задании инициализация необходима для корректного выполнения автопроверки.

3. *4-х разрядный счетчик Грея на основе ROM.

Счетчик должен считать примерно один раз в секунду, результат выводить на светодиоды, а по сигналу сброса устанавливаться в нулевое состояние. В ROM должны быть записаны выходные значения всех состояний счетчика Грея.
При выполнении задания использовать подход и решения из 4 задания 7 лабораторной работы, а также из 1 задания текущей лабораторной работы. Также, как и в задании 7.4 для автопроверки задействовать 1-й разряд счетчика, а для проверки работы на плате – 26-й.

Использовать следующий шаблон для объявления интерфейса модуля и его архитектуры:

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

entity lab73 is
    Port ( clk : in STD_LOGIC;
           srst : in STD_LOGIC;
           dout : out STD_LOGIC_VECTOR (3 downto 0));
end lab73;

...
...
--    en <= cntr(26) and (not cntr26d); -- For implementation
    en <= cntr(1) and (not cntr26d); -- For simulation
    delay_proc : process(clk) begin
        if rising_edge(clk) then
--            cntr26d <= cntr(26); -- For implementation
            cntr26d <= cntr(1); -- For simulation
        end if;
    end process;
...
-------------------------------------
```

4. **RAM 16 слов по 32 бита и тестбенч для нее. Без автопроверки.

Для задания разрядности данных и объема памяти использовать generic при объявлении интерфейса.

Тестбенч должен генерировать сигнал тактирования, счетчик, выполняющий счет до 8191 (шаг счета 8), бесконечный счетчик от 0 до 15 (шаг счета 1), сигнал разрешения записи, который первые 20 тактов равен 0, следующие 16 тактов равен 1 и оставшееся время до конца моделирования равен 0. Первый счетчик подключить к порту данных, второй – к адресному порту RAM. С помощью тестбенча необходимо проанализировать поведение разработанного устройства.

Также необходимо собрать проект и проверить тип занятых ресурсов. Убедиться в том, что для реализации памяти выбраны не таблицы истинности, а встроенная в FPGA блочная память (RAMB).

5. **Примените команды Tcl к любому из заданий 1-4.

   Найдите описание следующих команд Tcl в документе “Vivado Design Suite Tcl Command Reference Guide”: `report_control_sets` и `report_high_fanout_nets`. Примените их к любому из заданий 1-4. Опишите назначение команд их полученные результаты.

   Не забудьте открыть синтезированный или разведенный проекты перед выполнением команд Tcl.