

Лабораторная работа № 1 Исследование САПР Quartus II

Цель работы: изучение основных инструментов САПР Quartus II в процессе синтеза цифровых устройств на программируемых логических интегральных схемах (ПЛИС).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Объект и архитектура объекта. В VHDL каждое цифровое устройство представляется как объект проекта. Созданный объект затем может быть использован либо как самостоятельный цифровой модуль, либо как составная часть более сложного объекта в другом проекте более высокого уровня.

Описание объекта обязательно включает в себя две составляющих: объявление (декларацию) объекта и описание архитектуры объекта.

Объявление объекта описывает его внешний интерфейс, т.е. указывает, как объект проекта выглядит «снаружи», и содержит описание его входных и выходных линий – портов. Иными словами, декларация объекта определяет его как многополюсник: некий «черный ящик» с заданными входами и выходами, что позволяет описать связи объекта с внешними устройствами. Простейший синтаксис объявления объекта имеет вид:

```
entity <имя_объекта> is  
port  
(  
  <имя порта 1>: <вид сигнала> <тип сигнала>;  
  ...  
  <имя порта N>: <вид сигнала> <тип сигнала>  
)  
end <имя_объекта>;
```

Здесь и далее по тексту зарезервированные слова VHDL отмечены полужирным шрифтом.

Для имени объекта и имен портов могут быть использованы идентификаторы, допустимые в языке VHDL [1, С. 31, 2, С. 38; 4, 5]. Для лучшего понимания программы рекомендуется использовать в качестве идентификаторов осмысленные имена. Язык VHDL не чувствителен к регистру.

Вид сигнала обозначает направление его передачи относительно объекта и может принимать следующие значения:

in – входной сигнал;

out – выходной сигнал;

inout – двунаправленная линия, т.е. сигнал, являющийся и

входным, и выходным.

Все типы, которые указываются в описании портов, на момент объявления сущности (**entity**) объекта должны быть уже известны. Если это не стандартные, встроенные в VHDL типы, то они должны быть описаны до декларации **entity**. Как правило, описания таких типов выносятся в отдельный пакет или библиотеку, которые подключают к проекту выше по тексту программы относительно объявления сущности. Описания типов языка VHDL приводятся в [1, С. 34-53]. Основным типом VHDL является тип **std_logic**, описание которого заключено в пакете **std_logic_1164** библиотеки **ieee**.

Архитектура объекта представляет взгляд на устройство «изнутри», т.е. описывает логику работы объекта и показывает, каким образом формируются выходные сигналы. Синтаксис объявления архитектуры объекта имеет вид:

```
architecture <имя архитектуры> of <имя объекта> is
    <декларации>;
begin
    <параллельные операторы>;
end <имя архитектуры>;
```

Поскольку архитектура по своей сути описывает «поведение» устройства и его структуру, в качестве ее имени принято использовать имена *Behavior* или *Struct*.

В декларативной части архитектуры, располагающейся до ключевого слова **begin**, объявляются константы, сигналы, типы и подтипы пользователя, функции и процедуры пользователя, а также компоненты (другие объекты, используемые в проекте). Объявленные в теле архитектуры типы, сигналы, подпрограммы видимы только в пределах этой архитектуры.

Исполнительную часть архитектуры составляют параллельные операторы: процессы и параллельные присваивания. Термин «параллельные» означает, что все операторы в исполнительной части тела архитектуры выполняются одновременно и их взаимный порядок в программе не важен. В то же время хорошим стилем программирования считается такое расположение параллельных операторов, которое соответствует блок-схеме алгоритма, реализуемого в архитектуре.

Внутри архитектуры нельзя присвоить значение входному порту и нельзя считать значение выходного порта.

Сигналы и процессы в VHDL. Важными понятиями языка VHDL являются сигнал и процесс.

Сигналом называется программный элемент, который переносит

от одного процесса или объекта к другому процессу или объекту значение и вместе с ним – синхронизирующее воздействие. Сигнал объявляется синтаксической конструкцией вида:

signal <имя сигнала>: <тип сигнала>;

Для присвоения сигналу значения используется параллельный оператор присваивания «<=>». Например, для присвоения сигналу *w* типа *std_logic* значения логической единицы нужно записать:

w <=> '1';

Источник сигнала обозначают термином «драйвер» (driver).

Оператор процесса – это параллельный оператор, представляющий основу языка VHDL. Его упрощенный синтаксис:

process (<список чувствительности>)

<декларации>;

begin

<последовательные операторы>;

end process;

Список чувствительности представляет собой перечень сигналов, при изменении которых меняется логика работы синтезируемого объекта.

В декларативной части процесса (в отличие от архитектуры) не могут быть объявлены сигналы, поскольку они используются для обмена информацией между процессами. Нескольким присваиваниям одного сигнала внутри процесса соответствует один драйвер сигнала. Все процессы в программе выполняются параллельно. Процесс невозможно поместить в другой процесс, так как там есть место только для последовательных операторов. Нельзя присваивать одному сигналу значения в разных процессах, поскольку для него внутри программы может быть определен только один драйвер. Более подробная информация о сигналах и процессах приводится в [2, С. 41, 60-62; 4, 5].

Пример проекта на VHDL.

Пусть необходимо описать цифровое устройство, состоящее из логических элементов И, И-НЕ и ИЛИ (рис. 1). В начале программы нужно ввести в нее тип **std_logic**, подключив библиотеку **ieee** и ее пакет **std_logic_1164**. Тип **std_logic** содержит описание логических уровней VHDL-объекта.

library ieee; -- подключение библиотеки;

use ieee.std_logic_1164.all; -- подключение пакета.

Далее объявляется объект. В данном примере ему назначается

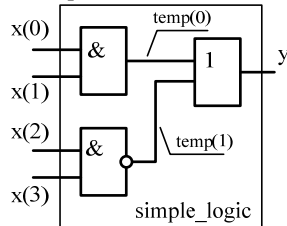


Рис. 1

имя *simple_logic*. Поскольку входные сигналы имеют одинаковый тип, удобно (для компактности описания) представить их в виде вектора.

```
entity simple_logic is
port
(
  x: in std_logic_vector (3 downto 0);
  y: out std_logic
);
end simple_logic;
```

Поскольку логические операции выполняются вне зависимости от изменения элементов входного вектора, оператор процесса в архитектуре можно не использовать. Для хранения промежуточных результатов логических операций И и И-НЕ введем сигнал *temp*.

```
architecture Behavior of simple_logic is
  signal temp: std_logic_vector (1 downto 0);
begin
  temp(0) <= x(0) and x(1);
  temp(1) <= x(2) nand x(3);
  y <= temp(0) or temp(1);
  -- если алгоритм работы простой, можно сразу записать:
  -- y <= ( x(0) and x(1) ) or ( x(2) nand x(3) );
end Behavior;
```

Компоненты. При структурном стиле программирования объект представляется в виде объединения более простых объектов. Допустим, для каждого из логических элементов на рис. 1 были составлены программы на языке VHDL, описывающие объекты AND_2, NAND_2 и OR_2. Чтобы «собрать» из них объект *simple_logic* (рис. 2), необходимо в декларативной части архитектуры объявить компоненты данных объектов. Основой для объявления компонента является сущность объекта: ключевое слово **entity** меняется на слово **component** и в конце пишется **end component**. Например, если сущность в программе для элемента 2-И была описана как

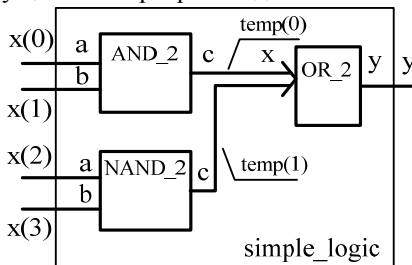


Рис. 2

```
entity AND_2 is
port
(
  a, b: in std_logic;
  c: out std_logic
);
end AND_2;
```

то описание компонента такого объекта будет иметь вид:

```

component AND_2 is
port
(
  a: in std_logic;
  b: in std_logic;
  c: out std_logic
);
end component;

```

Поскольку программы для NAND_2 и OR_2 могут писать разные программисты, то в результате для этих объектов могут получиться следующие компоненты.

```

component NAND_2 is
port
(
  a, b: in std_logic;
  c: out std_logic
);
end component;

```

```

component OR_2 is
port
(
  x: in std_logic_vector(1 downto 0);
  y: out std_logic
);
end component;

```

Для подключения к входам компонентов нужных сигналов в исполнительной части архитектуры могут использоваться два типа присвоения [2, С. 82-83; 4].

1. **При позиционном присвоении** важен порядок объявления входных и выходных сигналов – в строгом соответствии с порядком их объявления в компоненте:

<метка>: <имя компонента> **port map** (<список сигналов>);

На примере компонента AND_2 подключение портов при позиционном присвоении будет записано как

Comp_AND: AND_2 **port map** (x(0), x(1), temp(0));

Поскольку в описании компонента порты AND_2 указаны в последовательности *a*, *b*, *c*, то при компиляции программы к входному порту *a* будет подключен сигнал x(0), к порту *b* – сигнал x(1), а выходом компонента будет являться сигнал temp(0). Метка (в примере это «Comp_AND») является обязательной и может иметь произвольное (при соблюдении требований к идентификатору языка VHDL), но не совпадающее с идентификатором компонента имя.

2. **При поименованном присвоении** сигнал для каждого порта указывается в явном виде с использованием оператора «=>»:

```

<метка>: <имя компонента> port map
    (<имя порта> => <имя сигнала>,
    ...
    <имя порта> => <имя сигнала>);

```

При данном типе присвоения порядок сигналов в **port map** не важен. На примере компонента **NAND_2** подключение портов при поименованном присвоении будет записано как

```
Comp_NAND: NAND_2 port map (c => temp(1), b => x(2), a => x(3));
```

К входу компонента **OR_2**, как следует из его описания, должен быть подключен сигнал с типом **std_logic_vector(1 downto 0)**, поэтому при использовании позиционного стиля присвоения такое подключение запишется в виде:

```
Comp_OR: OR_2 port map (temp, y);
```

Запись «**Comp_OR: OR_2 port map (temp(1), temp(0), y);**» является ошибочной, так как она не соответствует описанию компонента **OR_2**.

Последовательностными называются цифровые устройства, выходные сигналы которых зависят не только от текущих значений входных сигналов, но и от последовательности значений входных сигналов, поступивших на входы устройства в предшествующие моменты времени. В языке VHDL работа последовательностных цифровых устройств, изменяющих свое состояние по переднему фронту тактового сигнала *clk*, задается в операторе процесса **process (clk)** синтаксической конструкцией вида:

```
if clk'event and clk = '1' then
```

```
<операторы>;
```

```
end if;
```

в которой атрибут **'event** указывает на момент изменения сигнала (выражение **clk'event** принимает значение **true** только в момент изменения сигнала *clk*), а уровень сигнала после его изменения определяет тип фронта: передний при **clk = '1'** и задний при **clk = '0'**. Подробное описание синтаксиса условного оператора **if** приведено в [2, С. 55; 4, 5].

Рассмотренная синтаксическая конструкция является основой для описания **D-триггеров** и устройств на их основе. VHDL-описание D-триггера с информационным входом *d*, тактовым входом *clk* и выходом *q* (все сигналы имеют тип **std_logic**) имеет вид [3, С. 116-119]:

```
process (clk)
```

```
-- триггер меняет свое состояние на значение,
```

```
-- действующее на информационном входе,
```

```
-- только в момент действия переднего фронта
```

```
if clk'event and clk = '1' then
```

```
q <= d;
```

```
end if;
```

```
end process;
```

Управляющими сигналами D-триггера ПЛИС типа FPGA являются вход разрешения тактовых импульсов *ena* и синхронный либо асинхронный входы сброса *reset* и установки *set*. Синхронный вход означает, что изменение состояния триггера (установка или сброс) произойдет только в момент прихода переднего фронта тактового импульса. Асинхронный вход означает, что изменение состояния триггера произойдет непосредственно в момент изменения сигнала на входе *set* или *reset*, и поэтому объявляется в списке чувствительности процесса.

- D-триггер с входами синхронного сброса,
- асинхронной установки и разрешения тактовых импульсов

```
process (clk, set)
begin
  if set = '1'
    then q <= '1';
    elsif clk'event and clk = '1'
      then
        if reset = '1'
          then q <= '0';
          elsif ena = '1'
            then q <= d;
          end if; -- завершаем условие if reset
        end if; -- завершаем условие if set
      end process;
```

Крайне нежелательно объединять условие действия фронта тактового сигнала (**clk'event and clk = '1'**) с другими условиями с помощью логических операторов **not**, **and**, **or**, **nand**, **nor** и **xor** во избежание синтеза неудачной для ПЛИС логической конструкции «*gated clock*» (тактовый сигнал, пропущенный через логический элемент), которая приводит к задержке фронтов тактового сигнала и трудно обнаруживаемым логическим ошибкам в работе синтезированной сущности.

Если в описании D-триггера заменить типы входа *d* и выхода *q* на **std_logic_vector** (*N*-1 **downto** 0), где *N* – натуральное число, большее единицы, то получится описание *N*-разрядного параллельного регистра.

На основе D-триггеров могут быть синтезированы такие последовательностные устройства, как счетчики и делители частоты.

Счетчиком называется последовательностное цифровое устройство, циклически меняющее свои состояния под действием импульсов, подаваемых на тактовый вход. Количество тактов, через которое повторяется исходное состояние счетчика, называют модулем счета *K_{сч}*. Счетчик содержит в своем составе регистр, хранящий его

текущее состояние. По фронту тактовой частоты значение регистра увеличивается (суммирующий счетчик) или уменьшается (вычитающий счетчик) на 1.

Для описания счетчика удобно ввести в оператор процесса переменную, которая хранит его текущее состояние.

Переменной в VHDL является объект, хранящий некоторое значение в пределах операторов процесса, функции или процедуры. Для объявления переменной используется ключевое слово **variable**.

variable <имя переменной>: <тип> := <начальное значение>;

Присвоение начального значения является необязательным. Декларация переменной допускается в процессе, процедуре или функции, где она используется. Переменная является локальной, т.е. область ее видимости в программе ограничена процессом, процедурой или функцией, внутри которых она определена. Присваивание значения переменной реализуется оператором присваивания « := »:

variable temp: **integer range** -128 to 127;

...

temp := temp + 2;

Присваивание переменной отличается от присваивания сигналу тем, что выполняется мгновенно [2, С. 52; 4]. Выполнение же оператора присваивания сигналу « <= > » внутри процесса означает только вычисление его значения и лишь назначение сигналу (так называемую транзакцию). Собственно присваивание сигналу нового значения фактически выполняется в момент остановки (завершения) процесса по ожиданию события. Рассмотрим это на примере.

process (clk)

variable var_x: **std_logic** := '0';

begin

if clk'event and clk = '1' **then**

var_x := '1';

sig_x <= '1';

if var_x = '1' and sig_x = '1' **then**

<операторы>;

else

<операторы>;

end if;

end if;

end process;

Пусть до начала выполнения процесса сигнал sig_x и переменная var_x типов **std_logic** имели нулевое значение. Тогда выполнение программы пойдет по ветви операторов **else**, так как на

момент вычисления условия ($\text{var_x} = '1'$ **and** $\text{sig_x} = '1'$) оператор сравнения $\text{var_x} = '1'$ вернет значение **true**, а оператор $\text{sig_x} = '1'$ – значение **false**, поскольку сигнал sig_x примет значение '1' только при завершении процесса **end process**.

Для описания суммирующего счетчика с модулем счета $K_{\text{сч}} = 12$ нужно определить сущность с тактовым входом clk и выходом counter типа **std_logic_vector** (3 **downto** 0), так как для представления числа «12» в двоичной системе счисления требуется 4 разряда. Процесс для реализации счетчика с использованием переменной имеет вид:

```
process (clk)
variable cnt: natural;
begin
if clk'event and clk = '1' then
    if cnt = 11 then cnt := 0;
    else cnt := cnt + 1;
    end if;
    -- преобразуем значение переменной cnt в 4-разрядный вектор
    -- с элементами типа std_logic [3, С. 112, 4, 5]
    counter <= conv_std_logic_vector (cnt, 4);
end if;
end process;
```

Для определения арифметической операции «+» к проекту нужно подключить пакет **std_logic.arith** библиотеки **ieee**, а для определения функции преобразования типов conv_std_logic_vector – пакет **std_logic_unsigned** из той же библиотеки.

Для реализации счетчика без использования переменной необходимо считывать его текущее состояние, поэтому выходной порт counter должен иметь тип **inout** [3, С. 119-120].

```
process (clk)
begin
if clk'event and clk = '1' then
    if conv_integer(counter) = 11
        then counter <= conv_std_logic_vector (0, 4);
        else counter <= counter + 1;
    end if;
end if;
end process;
```

Несколько усложнив программу для суммирующего счетчика, можно на ее основе составить программы для делителя и модулятора с широко-импульсной модуляцией (ШИМ).

Делитель с коэффициентом деления N предназначен для формирования из импульсного сигнала с частотой f_0 импульсного

сигнала с частотой f_0/N . Для формирования меандра (периодического импульсного сигнала со скважностью 2) *clk_out* с частотой f_0/N из тактового сигнала *clk* с частотой f_0 нужно сравнивать текущее значение счетчика с модулем счета N с величиной $N/2$: если оно меньше, то *clk_out* <= '0', в противном случае *clk_out* <= '1' (либо наоборот: сначала *clk_out* <= '1', а потом *clk_out* <= '0').

Для синтеза **параметрических** устройств используют настроечные константы. Настроечная константа **generic** задается в описаниях сущности и компонента до перечня портов **port** [1, С. 111-112; 2, С. 42; 4], например:

```
-- объявление N-разрядного параллельного регистра
-- значение по умолчанию для generic обязательно
entity Reg is
  generic (N: natural := <значение по умолчанию>);
  port (X : in std_logic_vector (N-1 downto 0);
        Y : out std_logic_vector (N-1 downto 0) );
end Reg;
```

Аналогичным образом настроечные параметры также декларируются перед портами компонентов. При использовании экземпляров компонентов перед **port map** нужно задавать требуемые значения настроечных параметров с помощью ключевых слов **generic map** (точка с запятой между **port map** и **generic map** не ставится).

```
-- вызов компонента «регистр» в проекте верхнего уровня
Register: Reg generic map (N => 16) port map (...);
```

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Синтез параметрического счетчика с асинхронным сбросом

1.1. Создать новый проект с именем *Counter* (П.1).

Проект, VHDL-файл верхнего уровня и сущность должны иметь одинаковое название (в данном случае – *Counter*).

1.2. Составить на языке VHDL описание счетчика с тактовым входом *clk*, входом сброса *rst* и восьмиразрядным (по умолчанию) выходом *Counter_out*.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
entity counter is
-- задаем настроечную константу
```

```

generic
(
N: natural := 8 -- коэффициент счета со значением по умолчанию
);
port
(
clk: in std_logic;
rst: in std_logic; -- сигнал сброса счетчика
counter_out: out std_logic_vector (7 downto 0)
);
end counter;

```

architecture Behavior of counter is

begin

-- алгоритм работы счетчика зависит от двух сигналов:

-- тактовых импульсов clk и сигнала сброса rst, поэтому

-- указываем их в списке чувствительности процесса

process(clk, rst)

-- вводим переменную для хранения текущего состояния

счетчика

variable cnt: **natural range** 0 to N-1 := <начальное значение>;

begin

-- реализуем асинхронный сброс, т.е. при активном уровне

-- сигнала сброса rst сразу же, не дожидаясь переднего фронта

-- тактовых импульсов clk, будем обнулять значение счетчика:

if rst = '1' **then**

cnt := 0;

-- для обнуления выходного порта используем агрегат

counter_out <= (**others** => '0');

-- иначе по переднему фронту

-- будем увеличивать значение счетчика

-- условие переднего фронта:

-- 1) сигнал изменился -> clk'event и (and)

-- 2) стал равным '1' -> clk = '1'

elsif clk'event and clk = '1' **then**

-- проверяем условие достижения максимального значения,

-- равного (Модуль счета -1)

if cnt = N - 1 **then**

cnt := 0;

else

cnt := cnt + 1;

```

end if; -- закрываем оператор if cnt = N - 1
counter_out <= conv_std_logic_vector(cnt, 8);
end if; -- закрываем оператор if с условием переднего фронта
end process;
end Behavior;

```

1.3. Откомпилировать проект. При наличии в тексте программы ошибок внести исправления.

1.4. Создать файл для моделирования работы счетчика, задав время моделирования – 20 мкс (**П.2**). Задать значения входных сигналов (**П.3**). Сигнал *clk* задать в виде меандра (периодического бинарного сигнала) с частотой 1 МГц, для чего в поле **Count Every** ввести значение половины периода – 500 нс. На линии *rst* в середине интервала моделирования задать короткий импульс высокого уровня, не совпадающий по времени с фронтами тактового сигнала.

По результатам моделирования убедиться в правильности работы проекта.

2. Синтез параметрического делителя частоты с выходным сигналом типа «меандр»

2.1. Составить на языке VHDL описание делителя частоты с задаваемым коэффициентом деления N (использовать настроечную константу **generic**). Для формирования сигнала типа «меандр» с частотой, в N раз меньшей частоты входных тактовых импульсов, нужно определить выходной сигнал типа **std_logic**, который зависит от состояния внутреннего счетчика, реализованного оператором процесса: если оно меньше $N/2$, выдавать логический «0», если больше – логическую «1».

2.2. Выполнить моделирование работы делителя для частоты тактовых импульсов $f_{CLK} = [N_n]$ МГц и $N = [2 * N_\phi]$ (N_n и N_ϕ – количество букв в имени и фамилии); определить максимальную тактовую частоту и задержку (**П.4**).

2.3. Изменить коэффициент деления таким образом, чтобы частота выходных импульсов составила 0,5, 1 или 2 Гц (на выбор), если входной высокочастотный тактовый сигнал имеет частоту 24, 27 или 50 МГц (на выбор)

2.4. Выполнить подключение выводов ПЛИС отладочного комплекта (**П.5**): тактовый вход подключить к выходу кварцевого генератора, а выход делителя – к красному или зеленому светодиоду (на выбор). Для определения номеров выводов ПЛИС при подключении руководствоваться описанием к отладочной плате *DE1 User Manual*. Откомпилировать проект заново.

2.5. Загрузить в ПЛИС созданный проект (**П.6**).

2.6. По частоте/периоду мигания светодиода сделать вывод о правильности работы синтезированного делителя частоты.

3. Синтез сущности из компонентов: каскадное соединение делителя и счетчика

3.1. Создать новый VHDL-проект.

3.2. Скопировать в рабочую директорию и подключить к проекту ранее созданные файлы делителя частоты и счетчика (П.8).

3.3. Объявить сущность с тактовым входом *clock*, входом сброса *reset* типа **std_logic**, выходным портом *Q* типа **std_logic_vector(7 downto 0)**.

3.4. Объявить в декларативной части архитектуры компоненты делителя и счетчика, а также сигнал *clk* типа **std_logic** для обозначения тактового сигнала с выхода делителя.

3.5. В содержательной части архитектуры описать подключение компонентов позиционным и/или поименованным способами согласно таблице и количеству букв в имени и фамилии, закодированным двумя битами: 0 – четное, 1 – нечетное.

Таблица

$N_n N_\phi$	Тип подключения компонента	
	Делитель	Счетчик
00	поименованный	поименованный
01	поименованный	позиционный
10	позиционный	поименованный
11	позиционный	позиционный

3.6. При подключении компонента «делитель» установить настроечную константу коэффициента деления, равную $10N_\phi$ для нечетного N_ϕ и $20N_\phi$ – для четного; на тактовый вход делителя подать *clock*, с выхода делителя снять сигнал *clk*.

3.7. Сигнал с выхода делителя подключить к тактовому входу компонента счетчика, а к его входу сброса подключить входной порт *reset*. Настроечную константу счетчика установить в зависимости от количества букв в имени N_n : равной $3N_n$ для нечетного N_n и $2N_n$ – для четного. К выходу счетчика подключить выходной порт сущности.

3.8. Откомпилировать проект. Создать файл моделирования.

3.9. Установить время моделирования 5 с, частоту входных тактовых импульсов *clock* – $10(N_n + N_\phi)$ Гц. Изменить систему счисления для представления выходных данных счетчика *Q* на беззнаковую десятичную (ПКМ на имени порта->**Properties->Radix->Unsigned Decimal**). По результатам моделирования сделать вывод о правильности работы составного проекта.

Приложение

П.1. Создание нового проекта

Выберите в меню **File->New->New Quartus II Project** и нажмите **OK**. В результате этих действий откроется диалоговое окно приглашения Мастера проектов **New Project Wizard: Introduction**, в котором перечислены основные шаги по созданию нового проекта. Навигация между этапами работы Мастера осуществляется кнопками «Next >» и «< Back».

1. Первым этапом работы Мастера (**Directory, Name, Top-Level...**) является определение каталога (директории) для размещения нового проекта (верхняя строка диалогового окна). По умолчанию этим каталогом является `c:\altera\90sp2\quartus`. Для изменения директории нажмите на кнопку с троеточием, расположенную справа от строки диалогового окна. Хорошим тоном считается создание для каждого нового проекта отдельной папки. В обозначении директории не должно быть символов кириллицы.

Далее во второй строке необходимо задать имя будущего проекта. По умолчанию он будет являться проектом верхнего уровня (*top-level design entity*), поэтому набираемое имя автоматически дублируется в третьей строке. Рекомендуется назначать проектам осмысленные имена, чтобы в дальнейшем иметь представление об их назначении. В имени проекта можно использовать только латинские символы, цифры и знак подчеркивания. Имя проекта не должно начинаться с цифры и знака подчеркивания.

2. На втором этапе работы мастера (**Add Files**) предлагается подключить к создаваемому проекту уже имеющиеся. **Если необходимо** подключить ранее созданный файл, кликните на кнопку с троеточием, в открывшемся диалоговом окне выберите файл с доступным расширением (*.vhd) и нажмите **Add**. Имя файла появится в прямоугольном поле под строкой ввода. Если файл был выбран ошибочно, его можно удалить, выделив имя мышкой и нажав **Remove**. Если файл подключать не требуется, нужно перейти к следующему этапу.

3. Следующий шаг – выбор микросхемы ПЛИС для реализации проекта (**Family & Device Settings**). Семейство ПЛИС выбирается в выпадающем списке в левом верхнем углу окна Мастера, а модель – в списке устройств семейства в нижней части окна. С использованием фильтров в правой части окна можно сократить количество отображаемых моделей ПЛИС, если заданы требуемый корпус (*package*), количество выводов (*pin count*) и производительность (*speed grade*).

Выберите семейство *Cyclone II* и ПЛИС *EP2C20F484C7*.

4. Следующий шаг – задание дополнительных настроек (**EDA Tool Settings**) – можно пропустить. Нажмите **Next**.

5. На заключительном этапе работы Мастера (**Summary**) отображаются результаты. Нажмите **Finish**. В результате в левом верхнем углу окна Quartus II в поле *Project Navigator* появится иерархическая структура созданного проекта в виде пиктограммы пирамиды желтого цвета с именем выбранной ПЛИС и подключенных к проекту файлов. По умолчанию к созданному проекту файлы на языке описания аппаратуры VHDL (*.vhd) не подключаются.

6. После создания проекта подключите к нему VHDL-файл: **File->New->VHDL File** (по умолчанию будет создан и открыт файл *Vhdl1.vhd*). Сохраните созданный файл **под тем же именем, что и имя проекта**, используя меню **File->Save As**. Убедитесь, что в поле **Add file to current project** (*Подключить файл к текущему проекту*) стоит галочка, и сохраните файл. По умолчанию этот файл будет файлом верхнего уровня в иерархии проекта (*Top Level Entity*).

Примечание: идентификаторы проекта, VHDL-файла верхнего уровня и объявленной в нем сущности **должны быть одинаковыми**.

П.2. Подключение файла моделирования

Выберите в меню **File->New->Vector Waveform File** и нажмите **OK**. Сохраните созданный файл с тем же именем, что и VHDL-файл верхнего уровня, используя меню **File->Save As**.

Кликните в левой части поля файла (под полем **Name**) правой клавишей мыши (ПКМ), в появившемся контекстном меню выберите пункт **Insert->Insert Node or Bus** (*вставить узел или шину*) и в открывшемся диалоговом окне нажмите кнопку **Node Finder** (*поиск узлов*). Проверьте, что в новом диалоговом окне в поле **Filter** выбрано значение «**Pins: all**». Это необходимо для отображения всех портов синтезированной сущности. Нажмите на кнопку **List** (*список*). Выделите (удерживая клавишу Ctrl и нажимая на левую клавишу мыши ЛКМ) из общего списка нужные шины и/или логические сигналы в левом поле и нажмите на кнопку «>>». Выбранные сигналы переместятся в правое поле. Если вы хотите переместить сразу все сигналы, нажмите кнопку «>>». Кнопки «<<» и «<<<» выполняют обратные действия. Нажмите **OK**.

Для задания необходимого интервала моделирования зайдите в меню **Edit->End Time** и, задав нужное численное значение в поле **Time** и суффикс (единицы времени) в выпадающем списке справа от данного поля, нажмите **OK**.


Чтобы видеть весь интервал моделирования, необходимо

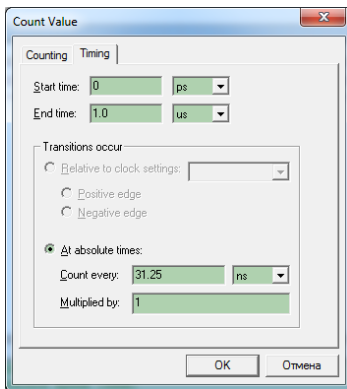
выбрать команду **View->Fit in Window** (сжать до пределов окна) или нажать горячую комбинацию клавиш **Ctrl + W**. Эту операцию также желательно повторять каждый раз после окончания моделирования (см. П.3).

Для просмотра значений интересующих внутренних переменных и сигналов их можно добавить вручную, записав в поле **Name** меню **Insert->Insert Node or Bus** имя сигнала или переменной. Если они не удалены компилятором в ходе оптимизации проекта, то в строке **Type** отобразится их тип, что будет означать возможность добавления сигнала/переменной в файл моделирования.

П.3. Моделирование работы объекта VHDL



Для перемещения интересующего сигнала выделите его однократным кликом ЛКМ по его имени, а затем, зажав ЛКМ, переместите вверх или вниз по списку.

Для задания периодических сигналов используется команда **Count Value** (пиктограмма ). В меню команды можно задать систему счисления в выпадающем списке **Radix** (как правило, при работе требуются двоичная **Binary**, шестнадцатеричная **Hexadecimal** и десятичные – со знаком и без – **Signed** и **Unsigned Decimal** системы счисления) и интервал изменения сигнала **Count every** (по умолчанию – 10 нс). Изменить интервал можно либо изменением значения в поле **Count every**, либо множителем в поле **Multiplied by**. Поле **Start Value** определяет начальное значение сигнала, а поле **Increment by** – величину, на которую увеличится сигнал на следующем интервале (для задания постоянного значения нужно ввести 0).

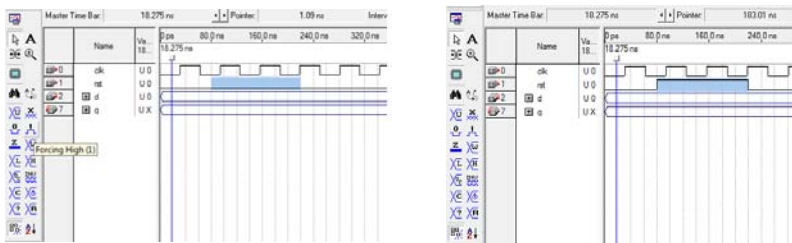



Для задания частоты тактового сигнала F нужно ввести в поле **Count every** значение половины периода этой частоты $0,5T = 1/(2F)$ с соблюдением суффикса. На рисунке ниже показан пример задания частоты тактового сигнала $F = 16$ МГц (полпериода $0,5T = 31,25$ нс).

Также задавать значения логического сигнала можно, выделяя требуемые интервалы ЛКМ и используя команды **Forcing High** (установка логической единицы) –

пиктограмма  или комбинация клавиш **Ctrl + Alt + 0** и **Forcing Low** (установка логического нуля) – пиктограмма  или комбинация

клавиш **Ctrl + Alt + 1**.



Для запуска моделирования нажмите пиктограмму .

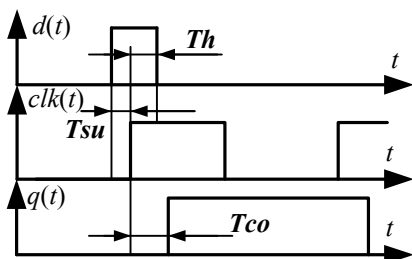
Для просмотра отдельных элементов шины нужно нажать на символ «+» слева от ее имени.

Для измерения временных интервалов используется курсор в виде квадратика, который по умолчанию устанавливается в конце файла симуляции. Для перемещения курсора нужно навести на него указатель мыши и зажать ЛКМ либо воспользоваться стрелками клавиатуры ← «влево» и → «вправо» либо дважды кликнуть ЛКМ на необходимом интервале под шкалой времени и после появления нового курсора сделать его основным, нажав на нем ПКМ и выбрав в контекстном меню **Make Master Time Bar**. При нажатии на стрелки курсор перемещается к следующему (по направлению его движения) моменту изменения входного/выходного сигнала. Установить курсор на требуемое модельное время можно с использованием пиктограммы **Go to** с изображением стрелки «→».

Для измерения времени задержки одного сигнала относительно другого курсор устанавливается на фронт первого сигнала, а указатель мыши передвигается к фронту второго сигнала. Интервал времени между курсором и указателем отображается в поле **Interval**. Другим способом измерения является ввод второго курсора, над которым указывается интервал времени до основного курсора.

П.4. Оценка задержек сигнала в проекте

Для определения величины задержек выберите пункт меню **Processing->Classic Timing Analyzer Tool**, в открывшемся окне нажмите кнопку **Start** и дождитесь окончания анализа. На вкладках **tpd**, **tsu**, **tco** и **th** наихудшие условия распространения сигнала будут отображены в верхней строке.



Tpd (*Time of propagation delay*) – время прохождения сигнала через комбинационную схему от входа до выхода.


Tco (*Time of clock-to-output*) – интервал времени от активного фронта тактового сигнала до появления сигнала на выходе.

Tsu (*Time of setup*) – интервал времени до активного фронта тактового сигнала, в течение которого на входе должен действовать установившийся логический сигнал.

Th (*Time of hold*) – интервал времени после активного фронта тактового сигнала, в течение которого на входе должен действовать установившийся логический сигнал.

Другим способом определения максимальной задержки является двойной клик на файле отчета **View Report** раздела **Classic Timing Analysis** в левой части окна Quartus II (окно **Tasks**). В нем максимальная задержка отображается в строке **Worst-Case** (наихудшие условия распространения).

П.5. Назначение выводов ПЛИС

Для задания соответствия сигналов проекта выводам выбранной ПЛИС выберите пункт меню **Assignments->Pins** или нажмите пиктограмму **Pin Planner** . В результате появится окно с топологией выводов выбранной ПЛИС и входными и выходными сигналами проекта.


Для назначения сигнала требуемому выводу нужно дважды кликнуть ЛКМ в строке с именем этого сигнала в поле **Location**, набрать имя вывода ПЛИС (можно без префикса **PIN_**) и нажать **Enter**. При назначении выводов ПЛИС отладочного комплекта следует использовать табличные данные из руководства **DE1 UserManual**. Проверьте, что в окне **Filter** выбрано значение «**Pins: all**». Это необходимо для отображения в **Pin Planner** всех выводов ПЛИС.

После закрытия **Pin Planner** результаты назначения выводов ПЛИС автоматически сохраняются. Подключение всех выводов ПЛИС, в том числе выводов питания и «земли», можно посмотреть в файле **<Имя проекта>.pin**, открыв его текстовым редактором (например, **WordPad** или **Notepad**).

Если в качестве идентификаторов портов синтезируемого

VHDL-объекта использовать имена, определенные производителем отладочного макета (приведены в файле *DE1_UserManual* на страницах 26-45), то можно выполнить автоматическое назначение выводов ПЛИС, загрузив файл с назначениями *DE1_TOP.qsf* в меню *Assignments->Import Assignments*.

П.6. Загрузка проекта в ПЛИС отладочной платы


Выберите пункт меню *Tools->Programmer* или нажмите пиктограмму .

В левом верхнем углу открывшегося диалогового окна нажмите на кнопку *Hardware Setup* и в новом диалоговом окне выберите программатор *USB-Blaster [USB-0]* из выпадающего списка.


Нажмите кнопку *Close*. Убедитесь, что в поле справа от кнопки *Hardware Setup* появилось имя выбранного программатора *USB-Blaster [USB-0]*.

Убедитесь, что переключатель *RUN/PROG* на отладочной плате находится в положении *RUN* (в верхнем положении).

Для загрузки программы на языке VHDL в ПЛИС используются файлы с расширениями **.pof* и **.sof* (так называемые «прошивки»), создаваемые при успешной компиляции проекта. По умолчанию программатор выбирает файлы с расширением **.sof*.

Для загрузки в ПЛИС созданного проекта выберите имя нужного файла «прошивки», нажмите кнопку *Start*  и дождитесь заполнения поля *Progress* до 100 % (занимает, как правило, не более 5 с).


П.7. Служебная информация

Вся информация о проекте сводится в файл отчета, просмотреть который после компиляции можно в меню *Processing->Compilation Report* или с помощью нажатия на пиктограмму .

В разделе отчета *Flow Summary* в строках *Total Logic Elements* и *Total Registers* приводятся абсолютные и относительные значения комбинационных функций и регистров ПЛИС, задействованных при синтезе проекта.

В разделе *Timing Analyzer->Clock Setup* в верхней строке приводится максимально допустимое значение тактовой частоты для синтезированного проекта.

В меню инструментов *Tools->Netlist Viewers->RTL Viewer* приводится схемотехническая реализация синтезированного устройства, а в меню *Tools->Chip Planner* или после нажатия пиктограммы  – физическое размещение на кристалле ПЛИС конфигурируемых логических блоков, задействованных при синтезе проекта. Для масштабирования изображения следует использовать

инструмент «лупа» : ЛКМ – увеличение, ПКМ – уменьшение. Последний инструмент также позволяет качественно оценить объем ресурсов ПЛИС, затраченных на реализацию проекта.

П.8. Подключение файлов к проекту

Выберите пункт меню **File->Open**, в открывшемся диалоговом окне поставьте галочку **Add file to current project**, затем выберите и откройте требуемый для подключения файл с расширением ***.vhd**.

Для просмотра и редактирования списка файлов, уже подключенных к проекту, используйте раздел меню **Project->Add/Remove files in project**.

Посмотреть содержимое подключенных к проекту файлов можно, нажав на пиктограмму в виде белого листа в нижней части панели **Project Navigator** (справа от пиктограммы пирамиды) и сделав двойной щелчок ЛКМ на интересующем файле.

Библиографический список

1. Суворова Е.А., Шейнин Ю.Е. Проектирование цифровых систем на VHDL. – СПб.: БХВ-Петербург, 2003. – 576 с.
2. Сергиенко А.М. VHDL для проектирования вычислительных устройств. – К.: ЧП «Корнейчук», 2003 – 208 с.
3. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. – М.: Горячая линия - Телеком, 2005. – 256 с.
4. <http://www.bsuir.by/vhdl/manual/>.
5. http://kanyevsky.kpi.ua/resourse/All/VHDL/VHDL_context.html.

Содержание

Лабораторная работа № 1 Исследование САПР Quartus II	1
Приложение.....	14
Библиографический список	20