Лабораторная работа 5

Синтаксические конструкции

В языке VHDL отсутствуют элементы для реализации триггеров и блоков памяти. Вместо этого для их построения используются специальные синтаксические конструкции. Другими словами, в зависимости от того, как выполняется присваивание сигналу, будет реализован тот или иной тип триггера.

Тактирование процессов

Заготовка для реализации синхронного цифрового устройства с синхронным сбросом:

```
process (clk) begin
      if rising_edge(clk) then
            if srst='1' then
            else
            end if;
      end if;
end process;
```

Описанный процесс запускается только по изменению сигнала тактирования, а все действия в процессе выполняются по его фронту. Все операции описываются внутри вложенного (второго) условного оператора.

Заготовка для реализации синхронного цифрового устройства с асинхронным сбросом:

```
process (clk,arst) begin
      if arst = '1' then
      elsif rising_edge(clk) then
      end if;
end process;
```

В приведенном примере сигнал сброса является асинхронным, т.е. процесс отреагирует на его изменение в любой момент времени, а не только в моменты фронтов сигнала тактирования, как это было в предыдущем примере.

Реализация триггеров

В описанном ниже процессе реализовано комбинационное устройство, т.к. для всех возможных значений входного сигнала enable указаны значения выходного сигнала Q. Элементы памяти в таком случае не требуются.

```
process (D,enable) begin
      if enable='1' then
            Q <= D;
      else
            Q <= '0';
      end if;
end process;
```

Пример реализации D-триггера-защелки представлен ниже. В указанном примере отсутствует спецификация значения выходного сигнала, в случае, когда сигнал enable не равен 1. В этом случае синтезатор считает, что значение сигнала Q не должно меняться. Т.е. необходим элемент памяти, а, т.к. нет сигнала синхронизации (нет синтаксической конструкции, описывающей динамическое управление), то будет реализован триггер-защелка.

```
process (D,enable) begin
```

```
      if enable='1' then
            Q <= D;
      end if;
end process;
```

Синтаксис синхронного D-триггера с динамическим управлением и входом разрешения работы представлен ниже. Здесь процесс запускается только при изменении сигнала clk (в списке чувствительности только один сигнал – clk), а синтаксическая конструкция `if rising_edge(clk) then` задает динамическое управление триггером. Также, как и в предыдущем примере, отсутствие альтернативной ветки в операторе условного выбора приведет к тому, что будет реализован элемент памяти.

```
process (clk) begin
      if rising_edge(clk) then
            if enable='1' then
                  Q <= D;
            end if;
      end if;
end process;
```

Для реализации синхронного триггера с асинхронными входами сброса и установки в списке чувствительности процесса необходимо указать сигнал тактирования и асинхронные сигналы управления. При этом в теле процесса во внешнем операторе условного выбора должны быть указаны необходимые асинхронные присваивания.

```
process (clk,clear,set) begin
      if clear='1' then
            Q <= '0';
      elsif set='1' then
            Q <= '1';
      elsif rising_edge(clk) then
            if enable='1' then
                  Q <= D;
            end if;
      end if;
end process;
```

В современных FPGA обычно физически реализованы D-триггеры (синхронные с динамическим управлением, с синхронными (reset/set) и асинхронными (clear/preset) входами сброса и установки), остальные типы триггеров строят на их основе.

Регистры сдвига

Заготовка для циклического сдвигающего регистра: shifter может быть как одноразрядным (сдвиг одноразрядных данных), так и многоразрядных (каждый элемент регистра представляет из себя слово с разрядностью больше 1). Во втором случае необходимо использовать составные типы данных, которые будут описаны в следующей лабораторной работе. В настоящей лабораторной работе все регистры оперируют одноразрядными данными.

```
process (clk) begin
      if rising_edge(clk) then
            shifter <= shifter (0) & shifter(7 downto 1);
      end if;
end process;
```

Заготовка для регистра сдвига (сдвиг вправо):

```
process (clk) begin
      if rising_edge(clk) then
            bit_out <= shifter(0);
```

```
            shifter <= bit_in & shifter(7 downto 1);
      end if;
end process;
```

Блоки задержки

Задержка сигнала на один такт:

```
process (clk) begin
      if rising_edge(clk) then
            delayed_signal <= input_signal;-- задержка сигнала на один такт
      end if;
end process;
```

Схема формирования строба из “длинных сигналов”. Эта схема потребуется при выполнении последнего подзадания следующей лабораторной работы.

```
signal data_valid       : std_logic; -- сигнал длительностью много тактов
signal data_valid_d     : std_logic;
signal strob            : std_logic;

process (clk) begin
      if rising_edge(clk) then
            data_valid_d <= data_valid;-- задержка сигнала на один такт
      end if;
end process;

strob <= data_valid and (not data_valid_d);
```

Задание к лабораторной работе 5

Реализовать синхронный JK-триггер, циклический сдвигающий регистр, блок задержки, генератор псевдослучайной последовательности и кодер сверточного кода. 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:

1. Синхронный JK-триггер с динамическим управлением, входом разрешения работы и асинхронными входами сброса и установки.
   Асинхронные входы и вход разрешения работы реализовывать на кнопках, J и K входы – на слайдерах. Подключить и корректно описать в XDC файле сигнал тактирования 100 МГц.

   Использовать следующий шаблон для объявления интерфейса модуля и архитектуры (в теле архитектуры, если необходимо, объявить внутренние сигналы выходов триггеров):

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

entity lab51 is
    Port ( clk    : in STD_LOGIC;
           clear  : in STD_LOGIC;
           preset : in STD_LOGIC;
           j : in STD_LOGIC;
           k : in STD_LOGIC;
           e : in STD_LOGIC;
           jk_out  : out STD_LOGIC);
end lab51;
-------------------------------------
```

2. 16-разрядный замкнутый (циклический) сдвигающий регистр и тестбенч к нему. Выполнить моделирование устройства.

   Инициализация регистра должна выполняться по сигналу сброса, сброс синхронный. Регистр должен иметь два входа: один – для сброса регистра в начальное состояние “1011111010101111”, второй – вход сигнала тактирования.

   Сдвиг выполнять слева направо (от самого старшего разряда (MSB) к самому младшему (LSB)).

   Тестбенч должен генерировать сигнал сброса и сигнал тактирования и включать в себя модуль регистра сдвига. На автопроверку подключать только регистр сдвига (тестбенч проверке не подлежит).

   Использовать следующий шаблон для объявления интерфейса модуля:

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

entity lab52 is
    Port ( clk : in STD_LOGIC;
           srst : in STD_LOGIC;
           dout : out STD_LOGIC_VECTOR (15 downto 0));
end lab52;
--------------------------------------
```

3. Блок задержки 8-разрядного сигнала на один такт и тестбенч к нему. Выполнить моделирование устройства.

   Без реализации на отладочной плате. Блок задержки должен иметь два входа (тактирования и информационный на 8 разрядов) и один 8-разрядный выход. Тестбенч должен генерировать сигнал тактирования и информационный сигнал. Информационный сигнал должен представлять из себя последовательность из трех различных чисел.

   Начальную инициализацию сигнала dout выполнять не нужно. Обратить внимание при моделировании, что в течение первого такта сигнал dout неопределен.

   Использовать следующий шаблон для объявления интерфейса модуля:

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

entity lab53 is
    Port ( clk  : in STD_LOGIC;
           din  : in STD_LOGIC_VECTOR (7 downto 0);
           dout : out STD_LOGIC_VECTOR (7 downto 0));
end lab53;
--------------------------------------
```

4. *Генератор ПСП скремблера системы WiMAX и тестбенч к нему. Выполнить моделирование устройства.

   Цифровое устройство должно иметь два входа: один – для сброса устройства в начальное состояние “101010001110110” (инициализация показана для MSB слева и LSB справа), второй – вход сигнала тактирования. Тестбенч должен генерировать сигналы сброса и тактирования.

   При реализации обратить внимание на следующий момент: по схеме генератора ПСП на рис. 6 лекции 7 на каждом такте рассчитывается исключающее ИЛИ между старшими двумя ячейками и отправляется одновременно на выход и в младший разряд регистра. При этом содержимое этого младшего разряда обновится (станет доступным) только на следующий такт, а результат исключающего ИЛИ доступен сразу. Другими словами результат исключающего ИЛИ старших разрядов регистра сдвига необходимо подавать в тактируемом процессе на младший разряд и вне процесса (т.е. без задержки в 1 такт) на выход. Вообще, вывода данных на выходной порт с комбинаторной логики, а не с выхода триггера (т.е. выходной сигнал присваивается в тактируемом процессе), лучше избегать.

   При моделировании сравнить генерируемую последовательность с ожидаемой. Первые биты ожидаемой последовательности (после снятия сигнала сброса): 1111100100…

Использовать следующий шаблон для объявления интерфейса модуля:

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

entity lab54 is
    Port ( clk : in STD_LOGIC;
           srst : in STD_LOGIC;
           psp_bit : out STD_LOGIC);
end lab54;
-------------------------------------
```

5. **Кодер сверточного кода (171,133) и тестбенч к нему. Выполнить моделирование устройства.

Разрабатываемое устройство состоит из двух компонентов: генератор ПСП из предыдущего подзадания (источник полезных данных) и, собственно, кодер.

Разрабатываемое устройство должно иметь входы сброса и тактирования и двухразрядный выход для кодированной последовательности.

Кодер должен иметь входы сброса, тактирования и информационный, а также один выход (двухразрядный): кодированная последовательность. Начальное состояние регистра сдвига кодера – все нули. Результат сложения по модулю 2 ячеек 171 отправлять в младший разряд выхода, результат сложения ячеек 133 – в старший разряд выхода.

В качестве информационной (кодируемой) последовательности использовать последовательность, генерируемую модулем, разработанным в предыдущем подзадании. Обратите внимание (!) на то, что в процессе работы кодера на каждый входной бит генерируется два выходных бита. Это значит, что разрядность информационного входа должна быть равна 1, а выхода – 2.

Тестбенч должен генерировать сигнал сброса и сигнал тактирования. Результатом выполнения подзадания являются корректные временные диаграммы работы кодера и прошивка FPGA.

Для проверки корректности работы кодера необходимо вручную посчитать значения первых трех пар канальных символов на выходе кодера для соответствующих входных информационных битов (по временной диаграмме симулятора).

Для успешного автотестирования все файлы компонентов устройства должны быть помещены в папку sources скрипта автопроверки.

Использовать следующий шаблон для объявления интерфейса модуля и архитектуры:

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

entity lab55 is
    Port ( clk  : in STD_LOGIC;
           srst : in STD_LOGIC;
           coded : out STD_LOGIC_VECTOR (1 downto 0));
end lab55;

architecture a of lab55 is
    signal info_bit  : std_logic;
    signal coder_reg : std_logic_vector(5 downto 0);

    component lab54 is
        Port ( clk     : in STD_LOGIC;
               srst    : in STD_LOGIC;
               psp_bit : out STD_LOGIC);
    end component lab54;
begin
...
-------------------------------------
```

Описание сигнала тактирования в XDC файле Artix7

```
set_property PACKAGE_PIN W5 [get_ports clk]

      set_property IOSTANDARD LVCMOS33 [get_ports clk]
      create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clk]
```