Оглавление

Процессы

Процесс в VHDL определяет независимую повторяющуюся последовательность операторов и представляет поведение некоторой части проекта. После того, как последний оператор последовательности выполнен, выполнение начинается с первого оператора процесса. Для определения процесса используется оператор process , который состоит из объявлений (после ключевого слова process) и операторной части, которая начинается после слова begin. В объявлении процесса допускается создавать переменные, в то время как объявлять сигналы в этой части нельзя.

Предложения в операторной части выполняются строго последовательно (в отличие от параллельного назначения потоковой формы описания и конкретизации компонентов в структурном стиле) и включают, подобно языкам высокого уровня, операторы присваивания, условные операторы и операторы циклов. Оператор "<=" здесь используется как оператор назначения сигнала (в отличие от потоковой формы, где он используется как оператор параллельного присваивания) и также выполняется последовательно.

Оператор процесса в VHDL существует в двух вариантах, которые определяют способ активации процесса (запуска на выполнение):

• вариант А — с помощью сигналов активизации:

```
PROCESS (X1, X2) — запускается только при изменении
 – хотя бы одного из сигналов X1, X2
 BEGIN
 Y <= not X1 and X2;
 Z <= Y or X3;
 END PROCESS;
```

• вариант Б – с помощью оператора WAIT задержки процесса, который может располагаться в любом месте операторной части процесса:

```
PROCESS (X1, X2)
 BEGIN
 Y <= not X1 and X2;
 Z <= Y or X3;
 WAIT ON X1, X2;
 END PROCESS;
 --или
 PROCESS (X1, X2)
 BEGIN
 WAIT ON X1, X2;
 Y <= not X1 and X2;
 Z <= Y or X3;
 END PROCESS;
```

Оператор WAIT в варианте Б позволяет задавать достаточно сложные условия активизации процессов:

WAIT ON <список сигналов> UNTIL <условие> FOR <время>

Например, для задания условий срабатывания схемы по фронту или по срезу сигнала C с задержкой на 10 ns или на t ns можно записать следующим образом:

```
WAIT ON C UNTIL C='1′ for 10 ns; — значение t предварительно
 – присваивается, например, t:=12 ns; или задается в GENERIC
 WAIT ON C UNTIL C='0′ for t ns;
```

Процессы в VHDL могут быть вложенными; также существует понятие пассивного процесса — процесса, в котором (явно или опосредовано) не производится назначение сигнала ни в одном из его операторов.

Поведенческий уровень описания объектов проекта является базовым, поскольку любые схемы, по крайней мере на самом нижнем уровне, представлены элементами, реализация которых выполнена на уровне поведения.

Структурный уровень описания объектов проекта. На структурном уровне объект представлен в виде иерархии связанных компонентов, на низшем уровне компоненты реализуются при помощи поведенческого описания. В структурном описании после объявления архитектуры следуют объявления компонентов, используемых в архитектуре. В теле описания архитектуры для создания экземпляров компонентов используются предложения конкретизации компонентов. Предложение конкретизации компонента начинается с метки, за которой следует имя компонента, а затем, если это необходимо, операторы назначения карты параметров (generic map) и карты портов (port map):

```
entity andnot2 — декларация имени объекта проекта
 port (x1, x2: in BIT; — декларация входных портов
 y: out BIT);— декларация выходного порта
 end andnot2;
```

```
architecture functional of andnot2 is — декларация архитектуры
 begin
 y <= not (x1 and x2) -- описание функции объекта
 end functional;

entity example is port (x1, x2, x3: in BIT; y: out BIT);
 end example;

architecture structure of example is
 component andnot2 is port (x1, x2: in BIT; y: out BIT);
 end component;

signal y1: BIT;
 begin
 P1: andnot2 port map (x1, x2, y1);
 P2: andnot2 port map (y1, x3, y);
 end structure;
```

Каждый экземпляр компонента должен быть сопоставлен с парой объект—архитектура. По умолчанию осуществляется поиск объекта с совпадающим с компонентом именем, списком портов и параметров, однако данное поведение можно изменить при помощи описания конфигурации.

Помимо конкретизации каждого компонента отдельно, существует возможность множественной конкретизации компонента при помощи оператора generic:

```
gen: for I in 1 to N generate
 P1: andnot2 port map (in1 (I), in2 (I), out (I));
 end generic;
```

В одном архитектурном теле допускается смешанное (mixed) описание, т.е. возможно одновременное использование процессной, потоковой и структурной форм, что в некоторых случаях значительно упрощает реализацию объекта.

Пример

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FIU is
```

```
GENERIC (
  CMD_CPU_W: integer:=32;                --Размер слова команды
  ADR_CPU_W: integer:=3;                 --Размер слова адреса
  FIFO_CPU_W:  integer:=4;               --Размер буфера
  IROM_ADR_W: integer:=8                 --Разрядность памяти команд

);

PORT(
        CLK:IN std_logic; --Синхронизация
        RST:IN std_logic; --Сигнал сброса
        RSTQ:IN std_logic; --Дополнительный сигнал сброса
        --Сигналы внешнего интерфейса
        CMD_CPU_VLD:IN std_logic; --Сигнал: «команда на шине данных валидна»
        ADR_CPU_IN:IN std_logic_vector(ADR_CPU_W-1 downto 0); --Адрес устройства на
шине
        CMD_CPU_IN:IN std_logic_vector(CMD_CPU_W-1 downto 0); --Код команды
        Q_CPU_FULL:OUT std_logic; --Очередь заполнена
        Q_CPU_EMPTY:OUT std_logic; --Очередь пуста
        --Интерфейс с процесосрным ядром
        CMD_SPU_START:OUT std_logic; --Сигнал запуска команды на исполнение
        SPU_RDY:IN std_logic; --Ядро готово к запуску команды
        CMD_SPU_CODE:out std_logic_vector(4 downto 0); --Код микрокоманды
        CMD_SPU_OP1:out std_logic_vector(31 downto 0); --Операны
        CMD_SPU_OP2:out std_logic_vector(31 downto 0); --
        CMD_SPU_OP3:out std_logic_vector(31 downto 0) --
);
end FIU;

architecture Behavioral of FIU is

        --CPU_FIFO_OUT interface
signal CMD_CPU_NXT: std_logic; --Impulse wdth must be 1 clock
signal ADR_CPU_OUT: std_logic_vector(ADR_CPU_W-1 downto 0);
signal CMD_CPU_OUT: std_logic_vector(CMD_CPU_W-1 downto 0);
signal Q_CPU_EMPTY_Int: std_logic;
```

```
	--SPU Instruction memory

constant DATA_WIDTH: integer:=80;
signal IROM_ADDR: std_logic_vector(IROM_ADR_W-1 downto 0);
signal IROM_DATA_OUT: std_logic_vector(DATA_WIDTH-1 downto 0);
signal IROM_DATA_OUT: std_logic_vector(DATA_WIDTH-1 downto 0);
signal DATA_FROM_FIFO,DATA_TO_FIFO: std_logic_vector(CMD_CPU_W+ADR_CPU_W-1
downto 0);

	--Регистр команды
constant CODE_WIDTH: integer:=5;

signal CMD_START,Q_CPU_EMPTYn: std_logic;

component FIFO_QUEUE is
GENERIC (
  DATA_W: integer;   --razrjadnost' slova komandi
  FIFO_W:      integer                    --Razmer FIFO
);
PORT(
	CLK:IN std_logic;
	RST:IN std_logic;
	RSTQ:IN std_logic;
	--IN interface
	DATA_VLD:IN std_logic; --Impulse wdth must be 1 clock
	DATA_IN:IN std_logic_vector(DATA_W-1 downto 0);
	Q_FULL:OUT std_logic;
	--OUT interface
	DATA_NXT:IN std_logic; --Impulse wdth must be 1 clock
	DATA_OUT:OUT std_logic_vector(DATA_W-1 downto 0);
	Q_EMPTY:OUT std_logic
);
end component;


component IROM is
```

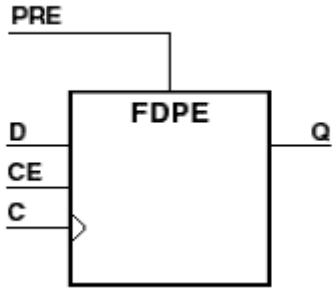
```
    GENERIC (
      ADDR_WIDTH : integer;
      DATA_WIDTH : integer;
      FILENAME : string
     );
    PORT(
      CLK : IN std_logic;
      RST : IN std_logic;
      ADDR : IN std_logic_vector(ADDR_WIDTH-1 downto 0);
      DATA : OUT std_logic_vector(DATA_WIDTH-1 downto 0)
      );
end component;


component CMD_DECODER is

    GENERIC (
      DATA_WIDTH : integer;
                        ADR_WIDTH : integer
     );
    PORT(
      CLK : IN std_logic;
      RST : IN std_logic;
                        CMD_CPU_VLD : IN std_logic;
                        CMD_CPU_NXT : OUT std_logic;
                        CMD_CPU_IN : IN std_logic_vector(31 downto 0);
                        TAG_CPU_IN : IN std_logic_vector(2 downto 0);
                        SPU_RDY : IN std_logic;
                        CMD_SPU_START : OUT std_logic;
      CMD_SPU_IN : IN std_logic_vector(DATA_WIDTH-1 downto 0);
                        CMD_SPU_CODE:out std_logic_vector(4 downto 0);
                        CMD_SPU_OP1:out std_logic_vector(31 downto 0);
                        CMD_SPU_OP2:out std_logic_vector(31 downto 0);
                        CMD_SPU_OP3:out std_logic_vector(31 downto 0);
                        IROM_ADR:out std_logic_vector(ADR_WIDTH-1 downto 0)
      );
```

```
end component;



begin

--CPU command FIFO

DATA_TO_FIFO<=ADR_CPU_IN&CMD_CPU_IN;
ADR_CPU_OUT<=DATA_FROM_FIFO(CMD_CPU_W+ADR_CPU_W-1 downto
CMD_CPU_W);
CMD_CPU_OUT<=DATA_FROM_FIFO(CMD_CPU_W-1 downto 0);

CPU_FIFO: FIFO_QUEUE
GENERIC MAP(
  DATA_W => CMD_CPU_W+ADR_CPU_W,
  FIFO_W => FIFO_CPU_W
)
PORT MAP(
        CLK=>CLK,
        RST=>RST,
        RSTQ=>RSTQ,
        --IN interface
        DATA_VLD=>CMD_CPU_VLD,
        DATA_IN=>DATA_TO_FIFO,
        Q_FULL=>Q_CPU_FULL,
        --OUT interface
        DATA_NXT=>CMD_CPU_NXT,
        DATA_OUT=>DATA_FROM_FIFO,
        Q_EMPTY=>Q_CPU_EMPTY_Int
);



--IROM
IROM_inst: IROM
GENERIC MAP(
```

```
        ADDR_WIDTH=>IROM_ADR_W,
        DATA_WIDTH=>DATA_WIDTH,
        FILENAME=>"irom.mem"
 )
PORT MAP(
        CLK=>CLK,
        RST=>RST,
        ADDR=>IROM_ADDR,
        DATA=>IROM_DATA_OUT
);



Q_CPU_EMPTYn<= not Q_CPU_EMPTY_Int;
Q_CPU_EMPTY<= Q_CPU_EMPTY_Int;

DECODER_Inst: CMD_DECODER

    GENERIC MAP (
      DATA_WIDTH=>DATA_WIDTH,
                        ADR_WIDTH=>IROM_ADR_W
     )
    PORT MAP(
      CLK=>CLK,
      RST=>RST,
                        CMD_CPU_VLD=>Q_CPU_EMPTYn,
                        CMD_CPU_NXT=>CMD_CPU_NXT,
                        CMD_CPU_IN=>CMD_CPU_OUT,
                        TAG_CPU_IN=>ADR_CPU_OUT(2 downto 0),
                        SPU_RDY=>SPU_RDY,
                        CMD_SPU_START=>CMD_SPU_START,
      CMD_SPU_IN=>IROM_DATA_OUT,
                        CMD_SPU_CODE=>CMD_SPU_CODE,
                        CMD_SPU_OP1=>CMD_SPU_OP1,
                        CMD_SPU_OP2=>CMD_SPU_OP2,
                        CMD_SPU_OP3=>CMD_SPU_OP3,
                        IROM_ADR=>IROM_ADDR(IROM_ADR_W-1 downto 0)
      );
```

end Behavioral;

Регистры




```
 Flip-Flop With Positive Edge Clock VHDL Coding Example
--
-- Flip-Flop with Positive-Edge Clock
--
library ieee;
use ieee.std_logic_1164.all;

entity registers_1 is
port(C, D : in std_logic;
Q : out std_logic);
end registers_1;

architecture archi of registers_1 is
begin
       process (C)
       begin
              if (C'event and C='1') then
                     Q <= D;
              end if;
       end process;
end archi;
```

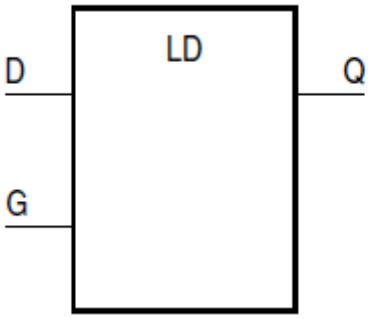
When using VHDL for a positive-edge clock, instead of using:

```
       if (C'event and C='1') then
```

you can also use:

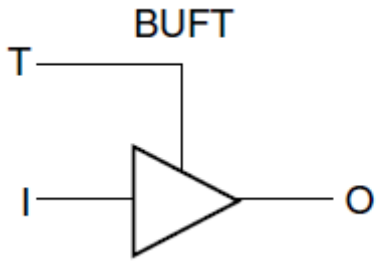
```
       if (rising_edge(C)) then
```

Latches (триггеры, управляемые уровнем)




```
--
-- Latch with Positive Gate
--
library ieee;
use ieee.std_logic_1164.all;

entity latches_1 is
port(G, D : in std_logic;
Q : out std_logic);
end latches_1;

architecture archi of latches_1 is
begin
	process (G, D)
	begin
		if (G='1') then
			Q <= D;
		end if;
	end process;
end archi;
```

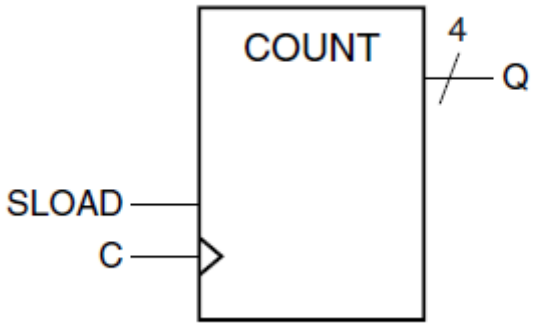
Буферы с третьим состоянием



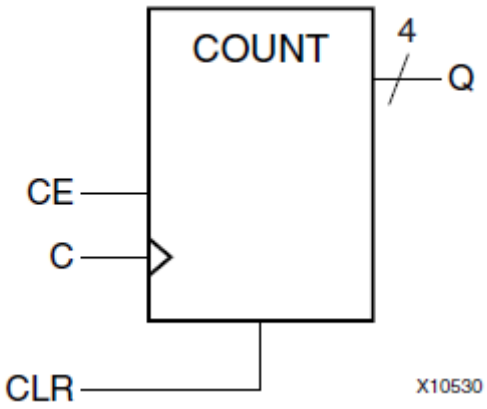

```
--
-- Tristate Description Using Combinatorial Process
--
library ieee;
use ieee.std_logic_1164.all;

entity three_st_1 is
port(T : in std_logic;
I : in std_logic;
O : out std_logic);
end three_st_1;

architecture archi of three_st_1 is
begin
	process (I, T)
	begin
		if (T='0') then
			O <= I;
		else
			O <= 'Z';
		end if;
	end process;
end archi;
```

Счетчики

Синхронная загрузка

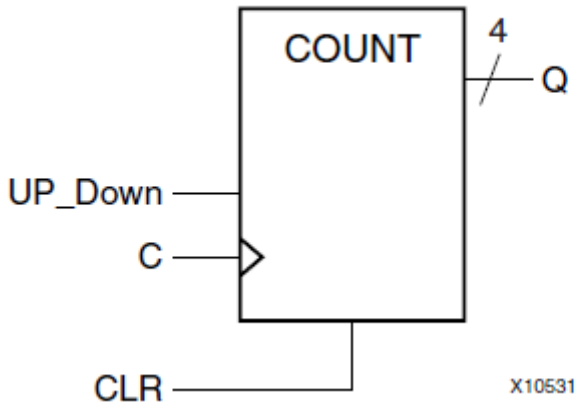



```
--
-- 4-bit Unsigned Up Counter with Synchronous Load with a Constant
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counters_4 is
port(C, SLOAD : in std_logic;
Q : out std_logic_vector(3 downto 0));
end counters_4;

architecture archi of counters_4 is
signal tmp: std_logic_vector(3 downto 0);
begin
	process (C)
	begin
		if (C'event and C='1') then
			if (SLOAD='1') then
				tmp <= "1010";
			else
				tmp <= tmp + 1;
			end if;
		end if;
	end process;
	Q <= tmp;
end archi;
```




```
if (CLR='1') then
tmp <= "0000";
elsif (C'event and C='1') then
```

```
if (CE='1') then
tmp <= tmp + 1;
end if;
end if;
```

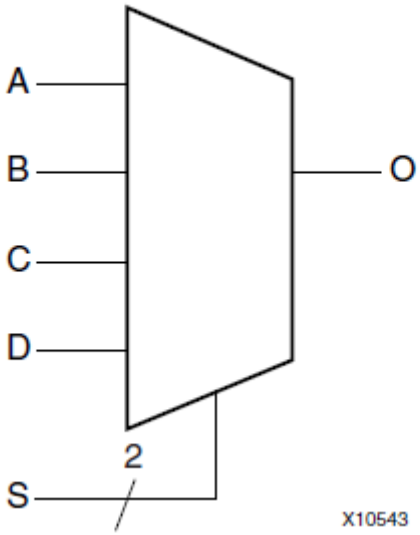



```
if (UP_DOWN='1') then
tmp <= tmp + 1;
else
tmp <= tmp - 1;
end if;
```

Сдвиговые регистры

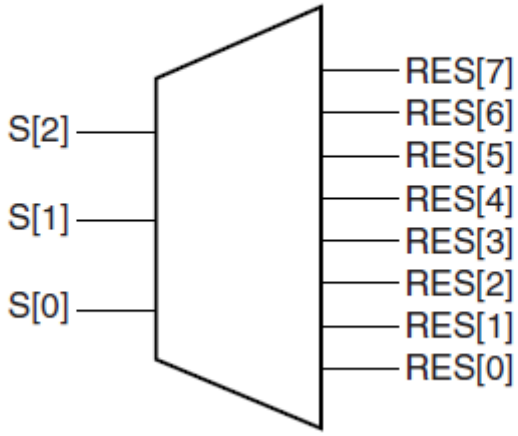
```
--
-- 8-bit Shift-Left Register with Positive-Edge Clock,
-- Serial In, and Parallel Out
--
library ieee;
use ieee.std_logic_1164.all;
entity shift_registers_5 is
port(C, SI : in std_logic;
PO : out std_logic_vector(7 downto 0));
end shift_registers_5;

architecture archi of shift_registers_5 is
signal tmp: std_logic_vector(7 downto 0);
begin
       process (C)
       begin
              if (C'event and C='1') then
                     tmp <= tmp(6 downto 0)& SI;
              end if;
       end process;
       PO <= tmp;
end archi;
```

Мультиплексоры




```
--
-- 4-to-1 1-bit MUX using an If statement.
--
library ieee;
use ieee.std_logic_1164.all;

entity multiplexers_1 is
port (a, b, c, d : in std_logic;
s : in std_logic_vector (1 downto 0);
o : out std_logic);
end multiplexers_1;

architecture archi of multiplexers_1 is
begin
	process (a, b, c, d, s)
	begin
		if (s = "00") then o <= a;
		elsif (s = "01") then o <= b;
		elsif (s = "10") then o <= c;
		else o <= d;
		end if;
	end process;
end archi;
```

Декодеры

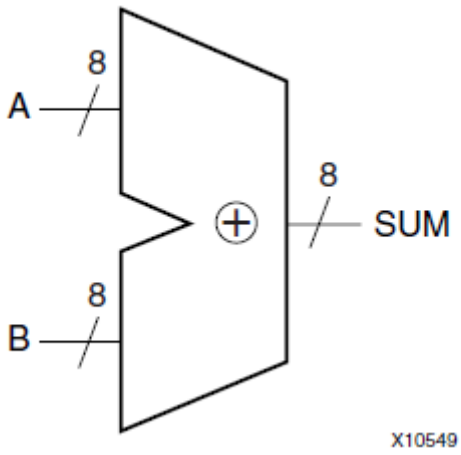



```
--
-- 1-of-8 decoder (One-Hot)
--
library ieee;
use ieee.std_logic_1164.all;

entity decoders_1 is
port (sel: in std_logic_vector (2 downto 0);
res: out std_logic_vector (7 downto 0));
end decoders_1;

architecture archi of decoders_1 is
begin
       res <= "00000001" when sel = "000" else
       "00000010" when sel = "001" else
       "00000100" when sel = "010" else
       "00001000" when sel = "011" else
       "00010000" when sel = "100" else
       "00100000" when sel = "101" else
       "01000000" when sel = "110" else
       "10000000";
end archi
```

Приоритетные шифраторы

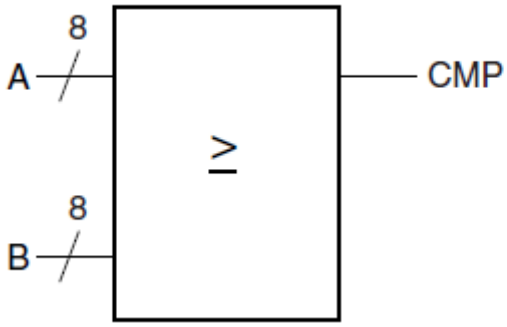
```
--
-- 3-Bit 1-of-9 Priority Encoder
--
library ieee;
use ieee.std_logic_1164.all;

entity priority_encoder_1 is
port ( sel : in std_logic_vector (7 downto 0);
code :out std_logic_vector (2 downto 0));

attribute priority_extract: string;
attribute priority_extract of priority_encoder_1: entity is "force";

end priority_encoder_1;

architecture archi of priority_encoder_1 is
begin
       code <= "000" when sel(0) = '1' else
       "001" when sel(1) = '1' else
       "010" when sel(2) = '1' else
       "011" when sel(3) = '1' else
       "100" when sel(4) = '1' else
       "101" when sel(5) = '1' else
       "110" when sel(6) = '1' else
       "111" when sel(7) = '1' else
       "---";
end archi;
```

Сумматоры




```
--
-- Unsigned 8-bit Adder
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adders_1 is
port(A,B : in std_logic_vector(7 downto 0);
SUM : out std_logic_vector(7 downto 0));
end adders_1;

architecture archi of adders_1 is
begin
      SUM <= A + B;
end archi;
```

Компараторы



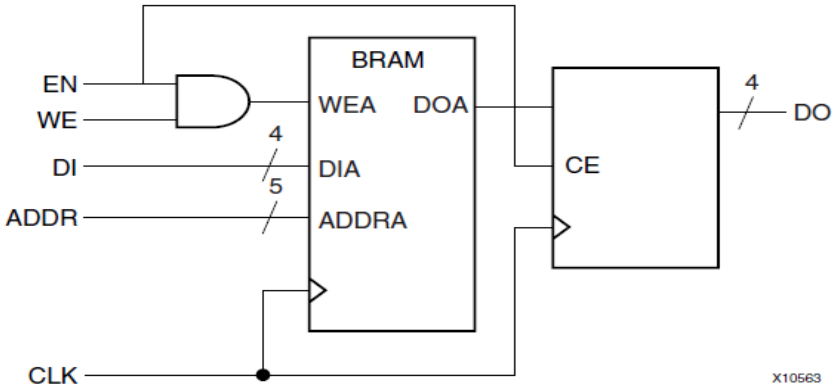

```
--
-- Unsigned 8-bit Greater or Equal Comparator
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity comparator_1 is
port(A,B : in std_logic_vector(7 downto 0);
CMP : out std_logic);
end comparator_1;

architecture archi of comparator_1 is
begin
       CMP <= '1' when A >= B else '0';
end archi;
```

Однопортовое ОЗУ

Типы ОЗУ в ПЛИС

• Synchronous write
• Write enable
• RAM enable
• Asynchronous or synchronous read
• Reset of the data output latches
• Data output reset
• Single, dual or multiple-port read
• Single-port/Dual-port write
• Parity bits (Supported for all FPGA devices except Virtex, Virtex-E, Spartan-II, and Spartan-IIE)
• Block Ram with Byte-Wide Write Enable
• Simple dual-port BRAM

Read-First RAM




```
--
-- Read-First Mode
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity rams_01 is
port (clk : in std_logic;
we : in std_logic;
en : in std_logic;
addr : in std_logic_vector(5 downto 0);
di : in std_logic_vector(15 downto 0);
do : out std_logic_vector(15 downto 0));
end rams_01;

architecture syn of rams_01 is
type ram_type is array (63 downto 0) of std_logic_vector (15 downto
0);
signal RAM: ram_type;
begin
       process (clk)
       begin
              if clk'event and clk = '1' then
```

```
            if en = '1' then
                  if we = '1' then
                        RAM(conv_integer(addr)) <= di;
                  end if;
                  do <= RAM(conv_integer(addr)) ;
            end if;
      end if;
end process;
end syn;
```