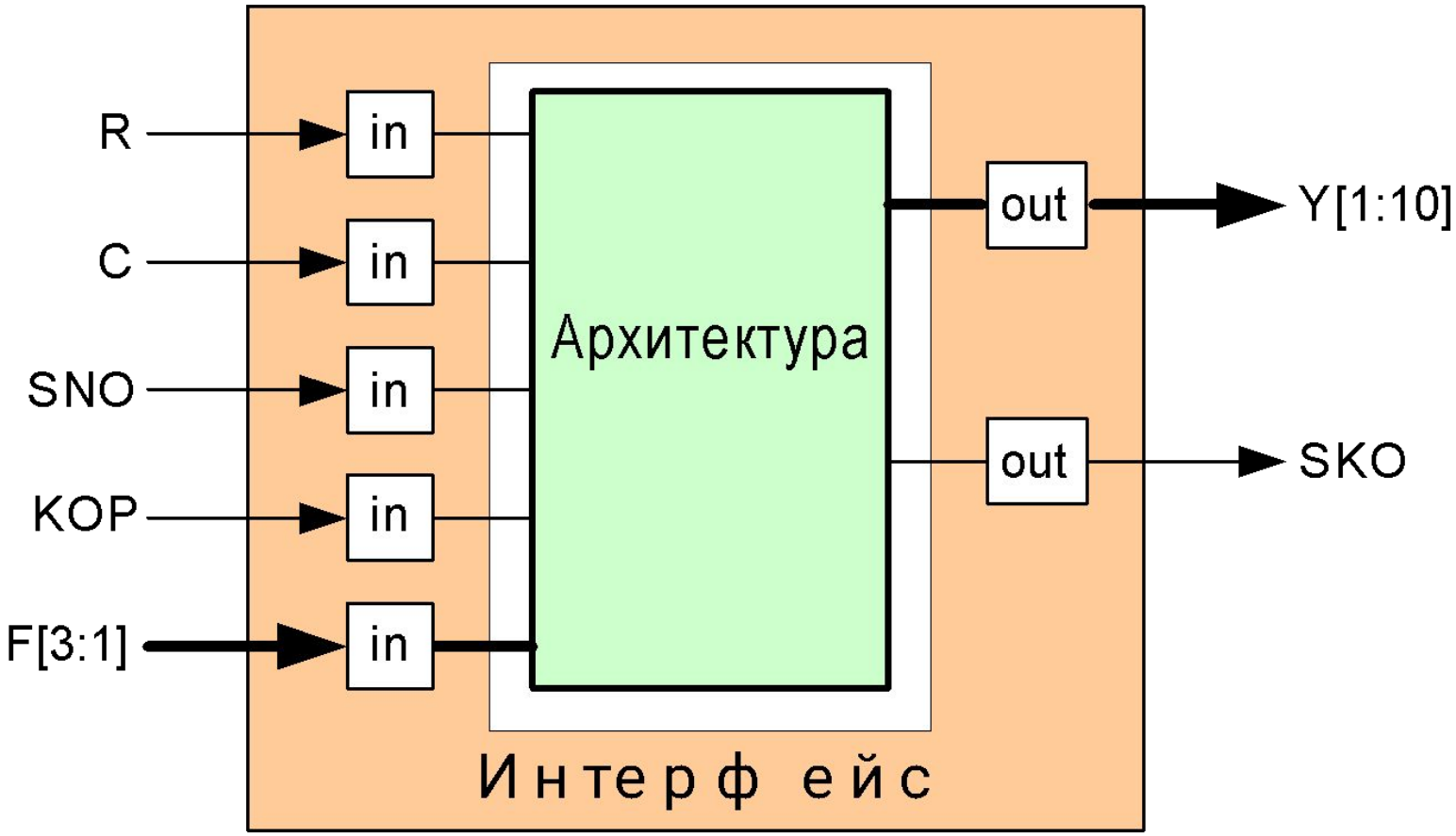
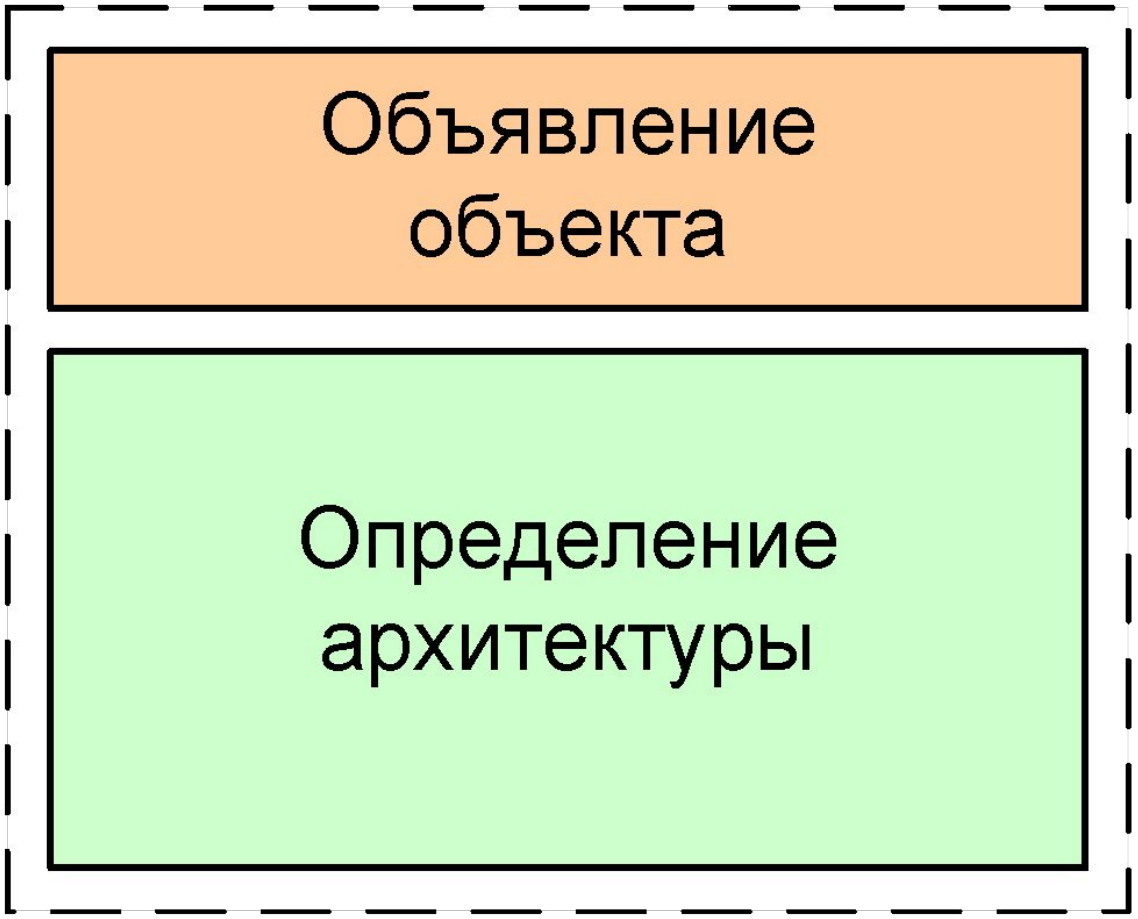

R
C
SNO
KOP
F[3:1]
in
in
in
in
in
Архитектура
out
out
Y[1:10]
SKO
Интерфейс

Текстовый файл (например, MUU.vhd)



Синтаксис объявления объекта на языке VHDL

```
entity  entity-name is
  port  (signal-names :  mode signal-type;
         signal-names :  mode signal-type;
         . . .
         signal-names :  mode signal-type);
end  entity-name;
```

Кроме ключевых слов **entity**, **is**, **port** и **end**, объявление объекта содержит следующие элементы:

entity-name	выбираемое пользователем имя объекта;	
signal-names	список выбираемых пользователем имен сигналов внешнего интерфейса, состоящий из одного имени или из большего числа имен, разделенных запятой;	
mode	одно из четырех зарезервированных слов, определяющих направление передачи сигнала:	
	in	сигнал на входе объекта;
	out	сигнал на выходе объекта; заметьте, что значение такого сигнала нельзя «прочитать» внутри структуры объекта; он доступен только объектам, использующим данный объект;
	buffer	сигнал на выходе объекта; в отличие от сигнала `out` его значение можно читать также внутри структуры данного объекта;
	inout	сигнал, который может быть входным или выходным для данного объекта; обычно этот режим используется применительно к входам/выходам схем с тремя состояниями;
signal-type	встроенный или определенный пользователем тип сигнала.	

Предопределенные синтезируемые типы данных языка VHDL

`bit`	`boolean`	`character`
`bit_vector`	`integer`	

Встроенные операторы для типов `integer` и `bit` приведены в табл. 1.3.

Таблица 1.3

Предопределенные операторы для типов `integer` и `bit` в языке VHDL

Операторы для типа `integer`		*Операторы для типа `bit`*	
`+`	сложение	`not`	инверсия
`-`	вычитание	`and`	И
`*`	умножение	`or`	ИЛИ
`/`	деление	`nand`	И-НЕ
`mod`	деление по модулю	`nor`	ИЛИ-НЕ
`rem`	остаток от деления по модулю	`xor`	ИСКЛЮЧАЮЩЕЕ ИЛИ
`abs`	абсолютное значение	`xnor`	ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ
`**`	возведение в степень		

Объявление объекта

```
entity MUU_125 is
    port (
        R, CLK, KOP, SNO: in BIT;
        F: in BIT_VECTOR (1 to 3);
        Y: out BIT_VECTOR (1 to 10);
        P: out BIT_VECTOR (3 downto 0);
        CS: out INTEGER range 0 to 3;
        SKO, TPO: out BIT
    );
end MUU_125;
```

О правилах записи программы

Как и в других языках программирования, в языке VHDL пробелы и переходы с одной строки на другую в общем случае игнорируются, и для удобства чтения их можно вставлять как угодно.

Комментарии начинаются с двух дефисов (--) и заканчиваются концом строки.

В языке VHDL определено много специальных строк символов, называемых *зарезервированными словами* или *ключевыми словами.* В приведенном примере имеется несколько ключевых слов: **entity, port, is, in, out, to, downto, range, end.**

Определяемые пользователем *идентификаторы* начинаются с буквы и содержат буквы, цифры и подчеркивания. Символ подчеркивания не может следовать за другим символом подчеркивания и не может быть последним символом идентификатора. В данном примере идентификаторами являются MUU_125, R, CLK, KOP, SNO, F, Y, P, CS, SKO, TPO. Зарезервированные слова и идентификаторы не чувствительны к регистру.

Синтаксис определения архитектуры

```
architecture architecture-name of entity-name is
type declarations
signal declarations
constant declarations
function definitions
procedure definitions
component declarations
begin
concurrent-statement
. . .
concurrent-statement
end architecture-name;
```

О*бъявление сигнала* сообщает ту же самую информацию о сигнале, какую содержит объявление порта, за исключением того, что вид сигнала (in, out, buffer, inout) не задается:

signal *signal-names : signal-type*;

Выделяют следующие стили проектирования и описания схем:

структурный,
потоковый,
поведенческий,
смешанный.

Элементы потокового проектирования

В потоковых проектах используют *параллельное присваивание*.

Параллельное присваивание определено в трех различных формах:

безусловное параллельное присваивание,
условное параллельное присваивание,
параллельное присваивание по выбору.

Синтаксис параллельных сигнальных операторов присваивания

```
signal-name <= expression;
signal-name <= expression when boolean-expression else
expression when boolean-expression else
. . .
expression when boolean-expression else
expression;
```

D0 = □A0 & □A1 & E; D1 = A0 & □A1 & E; D2 = □A0 & A1 & E; D3 = A0 & A1 & E.

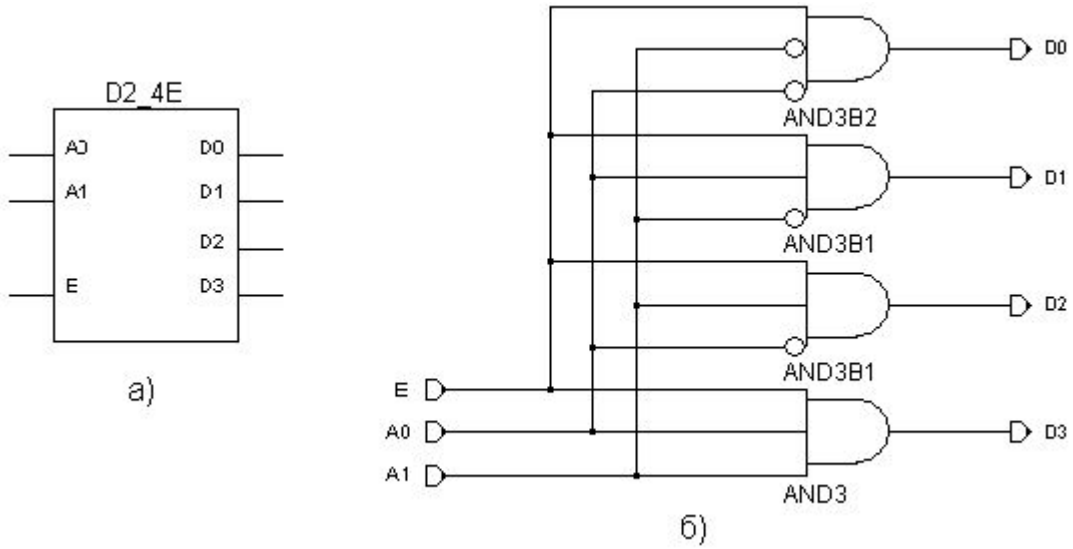


Рис. 1.3. Условное графическое обозначение (а) и логическая схема (б) дешифратора

Листинг 1.2. Потоковая архитектура для дешифратора

```
entity DECODER is
    port (
        A0,A1,E: in BIT;
        D0,D1,D2,D3: out BIT
    );
end DECODER;
architecture DECODER_arch of DECODER is
begin
  D0 <= not A0 and not A1 and E;
  D1 <=     A0 and not A1 and E;
  D2 <= not A0 and     A1 and E;
  D3 <=     A0 and     A1 and E;
end DECODER_arch;
```

Листинг 1.3. Потоковая архитектура для дешифратора

```
entity decoder is
    port (
        A: in BIT_VECTOR (1 downto 0);
        E: in BIT;
        D: out BIT_VECTOR (0 to 3)
    );
end decoder;
architecture decoder_arch of decoder is
begin
  D(0) <= not A(0) and not A(1) and E;
  D(1) <=     A(0) and not A(1) and E;
  D(2) <= not A(0) and     A(1) and E;
  D(3) <=     A(0) and     A(1) and E;
end decoder_arch;
```

Листинг 1.4. Архитектура дешифратора, в которой использованы условные присваивания

```
entity DECODER is
    port (
        A: in BIT_VECTOR (1 downto 0);
        E: in BIT;
        D: out BIT_VECTOR (3 downto 0)
    );
end DECODER;
architecture DECODER_arch of DECODER is
begin
  d(0) <= '1' when A="00" and E='1' else '0';
  D(1) <= '1' when A="01" and E='1' else '0';
  d(2) <= '1' when A="10" and e='1' else '0';
  D(3) <= '1' when A="11" and e='1' else '0';
end DECODER_arch;
```

Листинг 1.5. Архитектура мультиплексора, в которой использовано условное присваивание

```
entity MUX is
    port (
        D: in BIT_VECTOR (3 downto 0);
        A: in BIT_VECTOR (1 downto 0);
        F: out BIT
    );
end MUX;
architecture MUX_arch of MUX is
begin
  F <= D(0) when A="00"  else
       D(1) when A="01"  else
       D(2) when A="10"  else
       D(3);
end MUX_arch;
```

Синтаксис избирательного сигнального оператора присваивания

```
with expression select
signal-name <= signal-value when choices,
signal-value when choices,
. . .
signal-value when choices;
```

Листинг 1.6. Архитектура дешифратора, в которой используется присваивание
сигналу его значения по выбору

```
entity DECODER is
    port (
        A: in BIT_VECTOR (1 downto 0);
        E: in BIT;
        D: out BIT_VECTOR (3 downto 0)
    );
end DECODER;
architecture DECODER_arch of DECODER is
signal D_i:  BIT_VECTOR (3 downto 0);
begin
  with A select D_i <= "0001" when "00",
                       "0010" when "01",
                       "0100" when "10",
                       "1000" when "11";
  D <= D_i  when E = '1'  else "0000";
end DECODER_arch;
```

Листинг 1.7. Архитектура устройства для обнаружения простых чисел,
в которой используется присваивание сигналу его значения по выбору

```
entity KC is
    port (
        N: in BIT_VECTOR (2 downto 0);
        F: out BIT
    );
end KC;
architecture KC_arch of KC is
begin
  with N select
    F <= '1'  when "001" | "010" | "011" | "101" | "111",
         '0'  when others;
end KC_arch;
```

Листинг 1.8. Описание устройства для обнаружения простых чисел,
носящее поведенческий характер

```
entity KC is
    port (
        N: in INTEGER range 0 to 7;
        F: out BIT
    );
end KC;
architecture KC_arch of KC is
begin
    with N select
    F <= '1'  when 1 | 2 | 3 | 5 | 7,
         '0'  when others;
end KC_arch;
```

Синтаксис оператора
process

```
process  (signal-name, signal-name,   ..., signal-name) - список чувствительностей процесса
type declarations
variable declarations
constant declarations
function definitions
procedure definitions
begin
sequential-statement   -----
. . .                  ----- Последовательно выполняемые операторы
sequential-statement   -----
end process;
```

```
entity DECODER is
    port (A: in BIT_VECTOR (1 downto 0);
          E: in BIT;
          D: out BIT_VECTOR (0 to 3));
end DECODER;
architecture DECODER_arch of DECODER is
begin
  process (A,E)
    variable V: BIT_VECTOR (0 to 3); --- Другие процессы эту переменную не видят
  begin
    V(0) := not A(0) and not A(1) and E; --
    V(1) :=     A(0) and not A(1) and E; -- Выполняются последовательно, прямо на выход
    V(2) := not A(0) and     A(1) and E; -- присваивать нельзя! Иначе на выходе результаты
    V(3) :=     A(0) and     A(1) and E; -- будут появляться последовательно!!!
    D <= V;     -- Тут уже обычное присваивание
  end process;
end DECODER_arch;
```

Если работаем внутри процесса с переменной, объявленной в этом процессе, то используем оператор присвоения (:=)

Синтаксис оператора `if`

```
if boolean-expression then sequential-statement
end if;
if boolean-expression then sequential-statement
else sequential-statement
end if
if boolean-expression then sequential-statement
elsif boolean-expression then sequential-statement
. . .
elsif boolean-expression then sequential-statement
end if;
if boolean-expression then sequential-statement
elsif boolean-expression then sequential-statement
. . .
eisif boolean-expression then sequential-statement
else sequential-statement
end if;
```

if применяется только внутри процесса!

```
entity decoder is
    port (
        A: in BIT_VECTOR (1 downto 0);
        E: in BIT;
        D: out BIT_VECTOR (3 downto 0)
    );
end decoder;
architecture decoder_arch of decoder is
begin
  process (A,E)
  begin
        if E='0'   then D <= "0000";
     elsif A="00"  then D <= "0001";
     elsif A="01"  then D <= "0010";
     elsif A="10"  then D <= "0100";
     elsif A="11"  then D <= "1000";
    end if;
  end process;
end decoder_arch;
```

Синтаксис оператора `case`

```
case expression is
  when choices => sequential-statements
  . . .
  when choices => sequential-statements
end case;
```

Применяется внутри процесса!

Архитектура устройства для обнаружения простых чисел, в которой использован оператор `case`

```
entity KC is
    port (
        N: in INTEGER range 0 to 7;
        F: out BIT
    );
end KC;
architecture KC_arch of KC is
begin
  process(N)
  begin
    case N is
      when 1 |2 |3 | 5 | 7 => F <= '1';
      when others => F <= '0';
    end case;
  end process;
end KC_arch;
```

```
entity DECODER is
    port (
        A: in BIT_VECTOR (1 downto 0);
        E: in BIT;
        D: out BIT_VECTOR (3 downto 0));
end DECODER;
architecture DECODER_arch of DECODER is
begin
  process (A,E)
    variable V: BIT_VECTOR (0 to 3);
  begin
    case A is
      when "00" => V := "0001";
      when "01" => V := "0010";
      when "10" => V := "0100";
      when "11" => V := "1000";
    end case;
    if E='1' then D <= V;
    else D <= "0000";
    end if;
  end process;
end DECODER_arch;
```

signal_name'attribute_name

Например, предопределенный атрибут event ассоциируется с каким-либо сигналом, допустим, с сигналом CLOCK. Тогда атрибут запишется CLOCK'event. Этот атрибут имеет тип boolean со значением true, когда значение CLOCK изменилось, и значение false - в противном случае.

```
entity D_ff is
    port (
        clock, reset, D: in BIT;
        Q, QN: out BIT
    );
end D_ff;
architecture D_ff_arch of D_ff is
begin
  process (reset, clock)
   begin
     if reset='1' then Q <= '0' ; QN <= '1';
     elsif clock'event and clock='1' then Q <= D; QN <= not D;
     end if;
   end process;
end D_ff_arch;
```

D-триггер!!!

```
entity RG is
    port (
        clk, R: in BIT;
        Q: out BIT_VECTOR (3 downto 0)
    );
end RG;
architecture RG_arch of RG is
signal iQ: BIT_VECTOR (3 downto 0);
begin
  process (R,clk,IQ)
    begin
      if R='1' then  IQ <= "0001";
      elsif CLK'event and CLK='1' then
            IQ <= IQ(2 downto 0) & IQ(3);
      end if;
    Q <= IQ;
  end process;
end RG_arch;
```

Конкатенация (объединение строк). К одномерным массивам можно применять конкатенацию, в результате которой элементы правого операнда добавляются в конец левого. Например: "abc" & "df" = "abcdf"

```
entity counter is
    port (
        clk, r: in BIT;
        Q: buffer INTEGER range 0 to 7
    );
end counter;
architecture counter_arch of counter is
begin
  process (r,clk)
begin
   if r='1' then  Q <= 0;
   elsif CLK'event and CLK='1' then Q <= Q + 1;
   end if;
  end process;
end counter_arch;
```