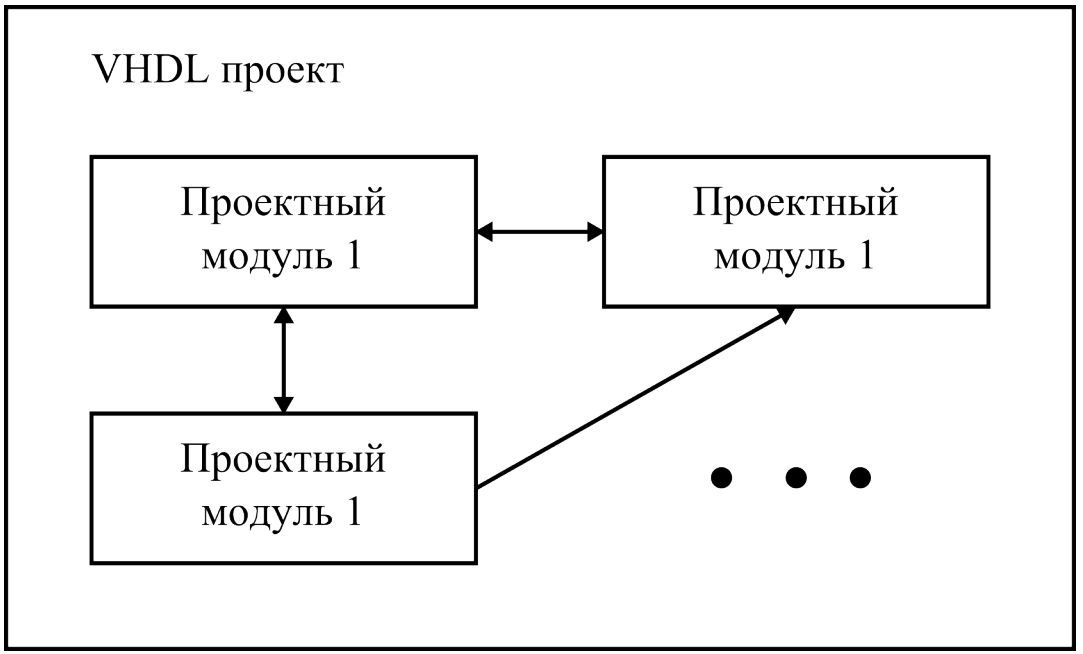
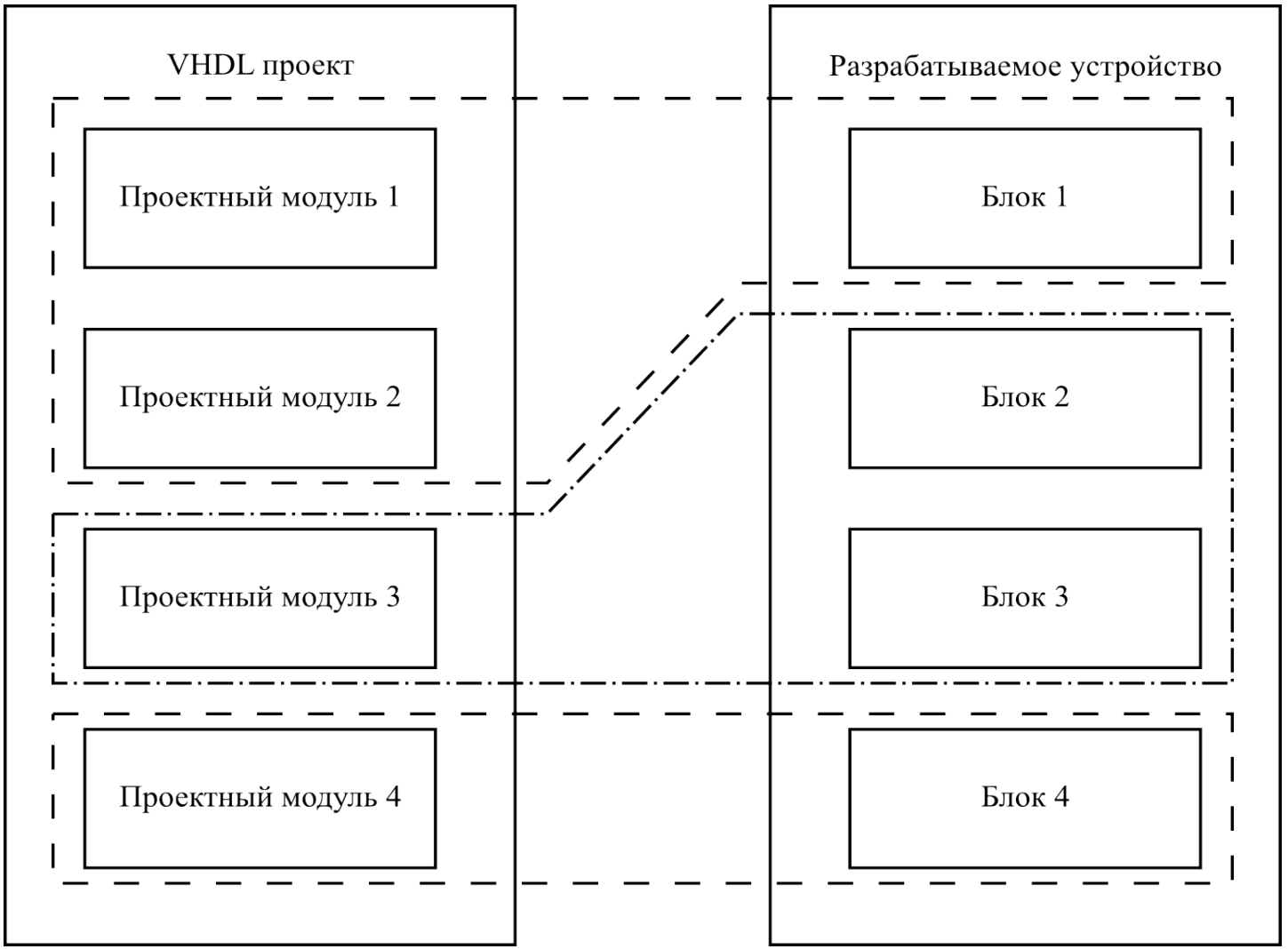
Проектные модули языка VHDL

Проект на языке VHDL представляет собой совокупность базовых элементов языка VHDL называемых ***проектными модулями***.

Проектные модули могут размещаться в одном файле или в разных файлах.



Взаимосвязь проектных модулей языка VHDL и блоков разрабатываемого устройства



Проектные модули языка VHDL

Первичные :

- **Entity (объект)**
- **Package (пакет)**
- **Configure (конфигурация)**

Вторичные :

- **Architecture (архитектура)**
- **Package body (тело пакета)**

Проектный модуль **Entity**

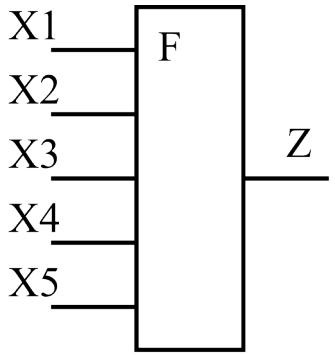
Entity (объект) – обязательный (в большинстве случаев) элемент VHDL проекта, предназначенный для спецификации интерфейса какого-либо объекта VHDL проекта. Позволяет определить входные и выходные порты объекта, а также различные параметры настройки, применяемые для модификации некоторых свойств объекта. При создании тестовых модулей, может создаваться объект не имеющий как выходных, так и выходных портов.

Проектный модуль **Entity**

Проектный модуль **Entity** определяет :

- Имя объекта
- Порты ввода/вывода объекта (не обязательный)
- Настроечные константы (не обязательный)
- Дополнительную информацию (не обязательный)

Проектный модуль **Entity** (пример)

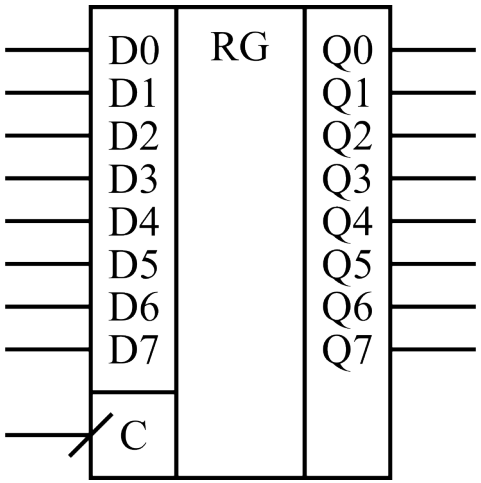


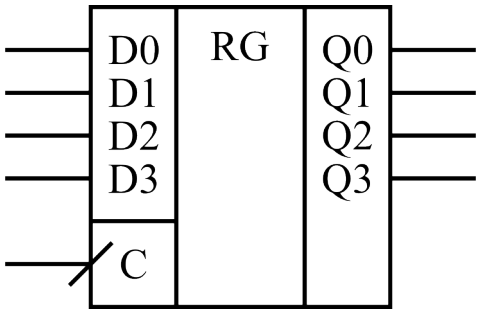
$$Z = \overline{X1 \cdot X2 + X3 + X4 \cdot X5}$$

```
entity F is
    port
    (
        X1,X2,X3,X4,X5      : in std_logic;
        Z                   : out std_logic
    );
end F;
```

```
entity F is
    port
    (
        X: in std_logic_vector(4 downto 0);
        Z: out std_logic
    );
end F;
```

Проектный модуль **Entity** (пример)






```
entity RG is
    generic
    (
        width : integer := 8
    );
    port
    (
        D :in  std_logic_vector( width-1 downto 0 );
        Q :out std_logic_vector( width-1 downto 0 )
    );
end RG;
```

Проектный модуль **Entity** (синтаксис)

```
entity имя_объекта is
    [настроечные константы объекта]
    [порты объекта]
    [декларативная часть]
[begin
    операторная часть]
end [entity] [имя_объекта] ;
```

Проектный модуль **Entity** (блок настроечных констант)

Блок настроечных констант определяет константы, которые могут использоваться при описании объекта, как в самом проектном модуле **entity** так и в проектном модуле **architecture**. Значения этих констант могут устанавливаться при создании экземпляра объекта.

Синтаксис :

```
generic
(
    имя_константы : тип_константы [:= значение_по_умолчанию];
    ...
    имя_константы : тип_константы [:= значение_по_умолчанию]
);
```

Проектный модуль **Entity** (блок портов)

Блок портов определяет интерфейс взаимодействия объекта с внешней средой, который в свою очередь определяется набором сигналов блока.

Синтаксис :

```
port
(
    имя_сигнала : режим_порта тип_порта ;
    ...
    имя_сигнала : режим_порта тип_порта
);
```

Проектный модуль **Entity** (блок портов)

Основные режимы работы порта :

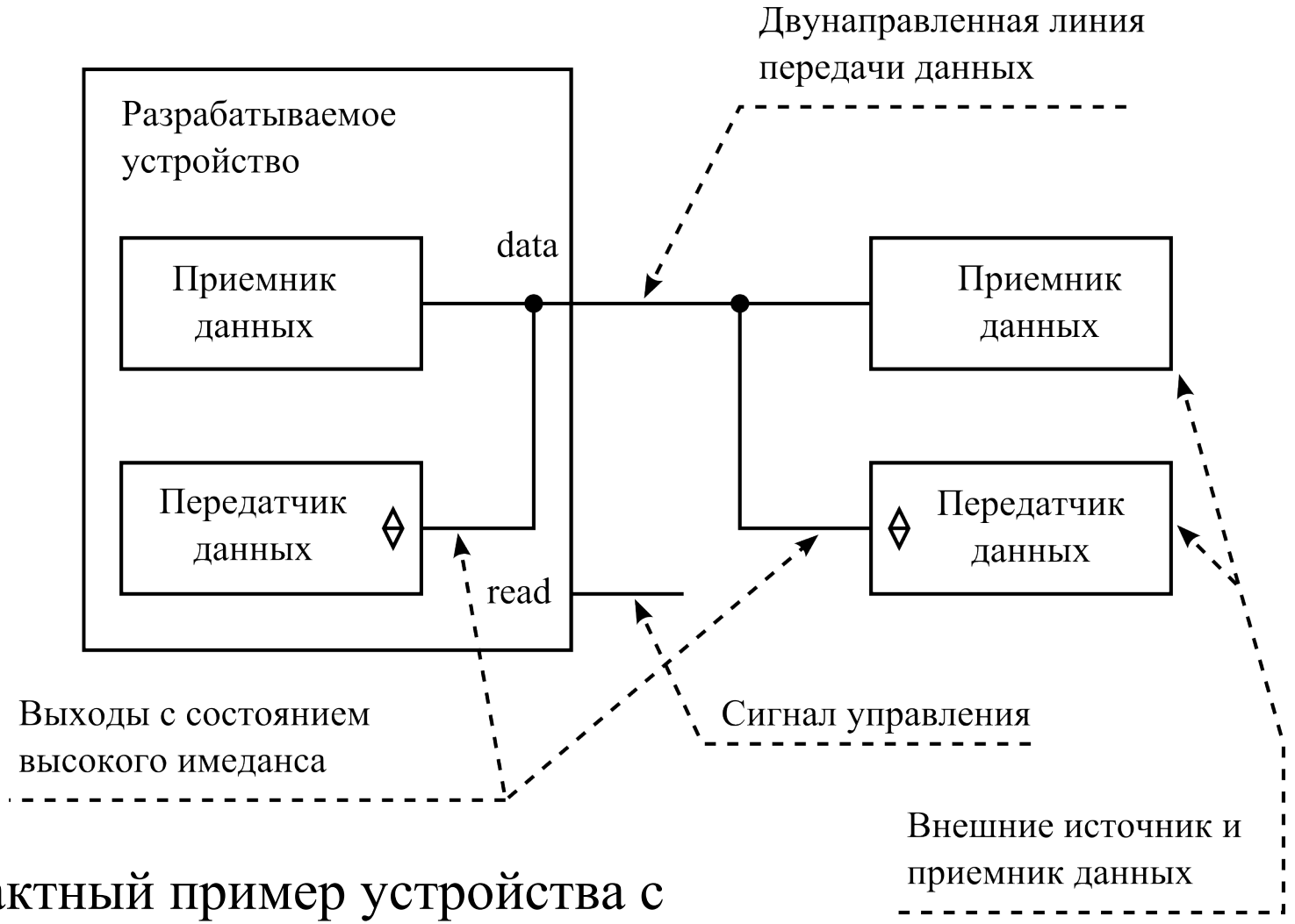
- **in** – данный сигнал (порт) является входным для объекта. Его можно только читать. Присвоение значений данному сигналу запрещено.
- **out** – данный сигнал (порт) является выходным для объекта. Ему можно только присваивать значения. Чтение данного сигнала запрещено.
- **inout** – данный сигнал является двунаправленным. Для этого типа порта разрешены операции как чтения так записи.

Проектный модуль **Entity** (блок портов)

Правила применения различных типов портов :

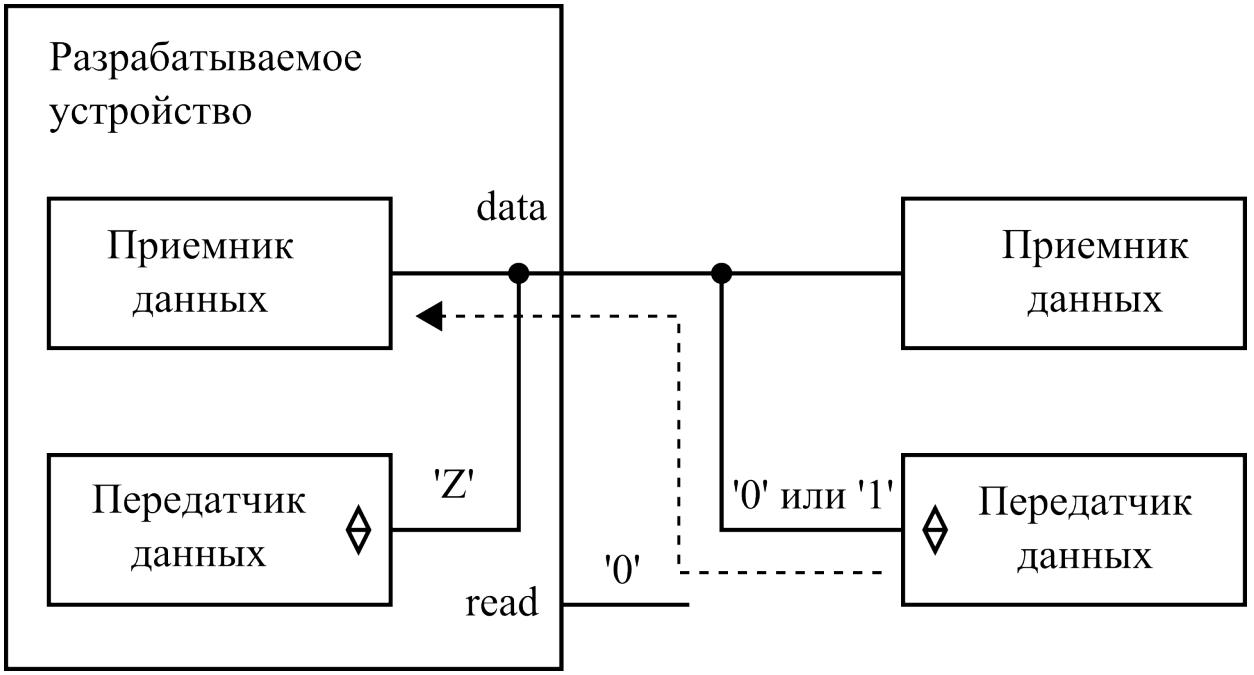
- тип **in** использовать для ***входных*** сигналов.
- тип **out** использовать для ***выходных*** сигналов.
- тип **inout** использовать ***исключительно*** для двунаправленных сигналов.
- не использовать тип **inout** для выходных сигналов, значения которых нужно считывать внутри объекта, определяемого проектным модулем **entity**. Для этого можно использовать сигналы, определенные внутри объекта.

Применение типа **inout** для двунаправленных портов



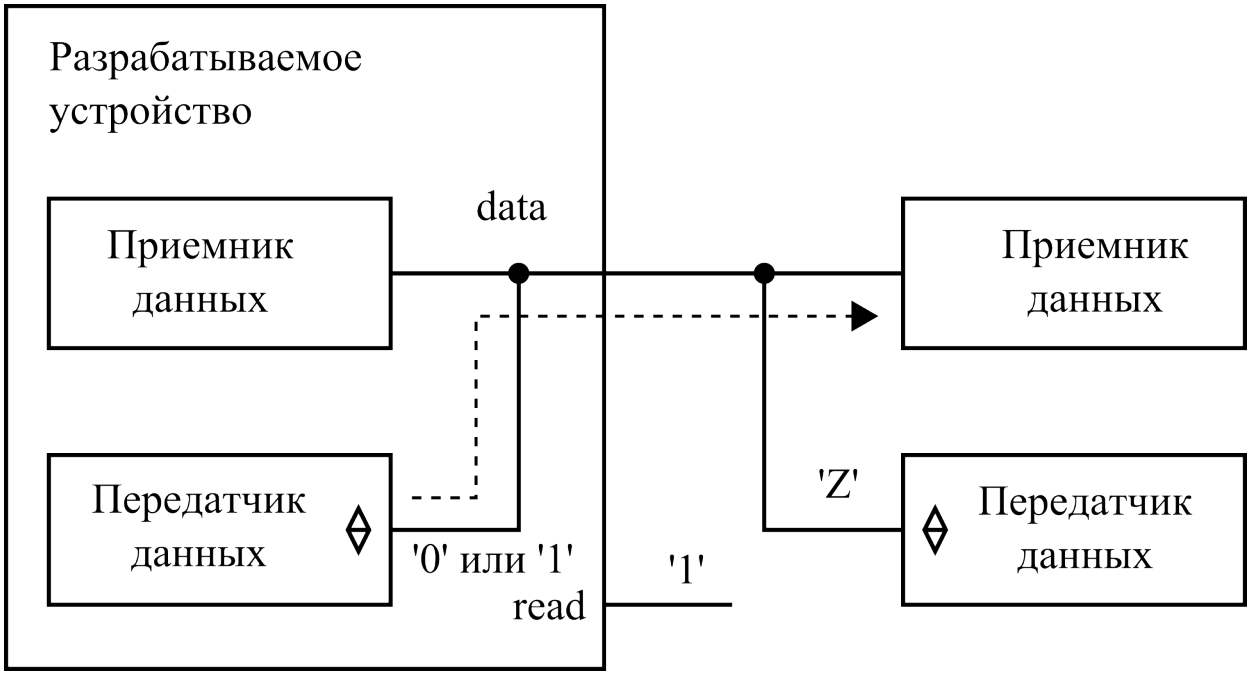
Абстрактный пример устройства с двунаправленными портами

Применение типа **inout** для двунаправленных портов



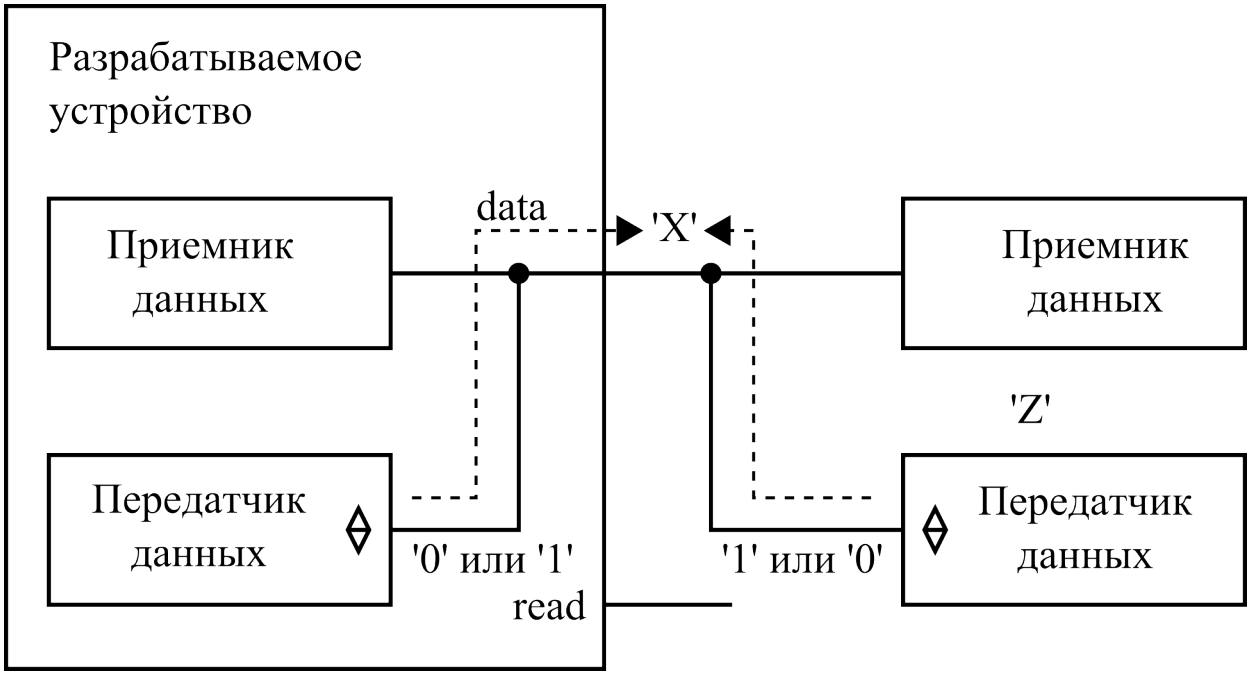
Режим записи в разрабатываемое устройство

Применение типа **inout** для двунаправленных портов



Режим чтения из разрабатываемого устройства

Применение типа **inout** для двунаправленных портов



Ситуация ошибки при использовании двунаправленных портов

Применение типа **inout** для двунаправленных портов

```
entity ff is
    port
    (
        data    :inout std_logic;
        read    :in std_logic
    );
end ff;

architecture behavioural of ff is

    signal data_int : std_logic;
    signal data_ext : std_logic;

begin

    data <= data_int when read = '1' else 'Z';
    data_int <= '1';
    data_ext <= data when read = '0';

end behavioural;
```

Проектный модуль **Entity** (декларативная часть)

Декларативная часть проектного модуля **entity** используется для определения различных объектов языка VHDL общих для объекта, определенного данным проектным модулем. Здесь могут быть определены :

- типы и подтипы данных ;
- подпрограммы ;
- константы ;
- атрибуты ;
- и т. д.

Чаще всего данные объекты используется в проектном модуле **architecture**

Проектный модуль **Entity** (операторная часть)

Операторная часть проектного модуля **entity** содержит набор операторов, общий для всех реализаций объекта, определенного данным интерфейсом.

Операторная часть может содержать только параллельные операторы.

Чаще всего используется для целей отладки разработки и включает операторы проверки каких либо условий, возникающих в объекте разработки в ходе его работы.

Развёрнутый пример объявления проектного модуля **Entity**

Настроечные константы

Порты ввода/вывода

Декларативная часть

Операторная часть

```
entity register is
    generic (
        width : integer := 8
    );
    port (
        input : in std_logic_vector(width-1 downto 0);
        output : out std_logic_vector(width-1 downto 0)
    );
    constant setup : time := 10 ns;

begin
    assert check = '1'
        report "Fatal error"
        severity failure;
end register;
```

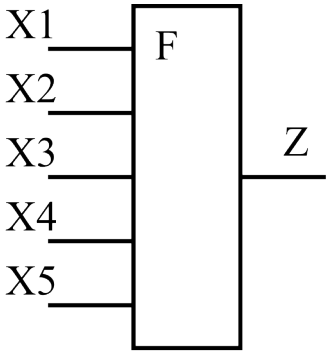
Проектный модуль **Architecture**

Проектный модуль **Architecture** определяет функциональность объекта, определенного проектным модулем **Entity**, и содержит операторы, определяющие его работу.

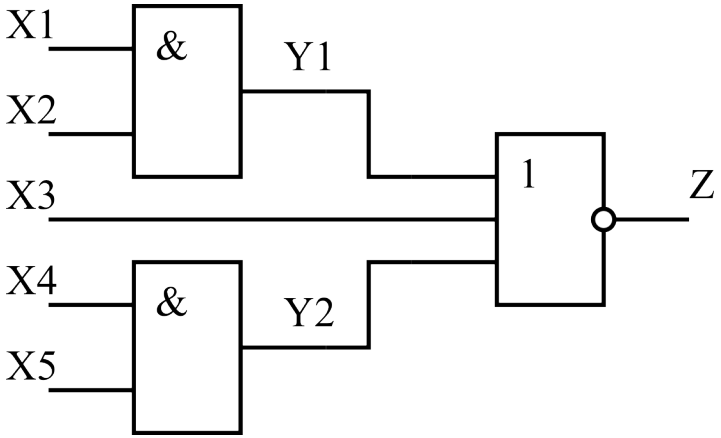
Основные свойства проектного модуля **Architecture** :

- обязательно принадлежит к какому-либо объекту, то есть к какому-либо проектному модулю **Entity**.
- одному проектному модулю **Entity** может принадлежать несколько проектных модулей **Architecture**.

Проектный модуль **Architecture**



$$Z = \overline{X1 \cdot X2 + X3 + X4 \cdot X5}$$




```
entity F is
    port
    (
        X1,X2,X3,X4,X5 : in std_logic;
        Z              : out std_logic
    );
end F;

architecture behavioural_1 of F is
begin
    Z <= not((X1 and X2)or(X4 and X5) or X3);
end behavioural_1;

architecture behavioural_2 of F is
    signal Y1 : std_logic;
    signal Y2 : std_logic;
begin
    Y1 <= X1 and X2;
    Y2 <= X4 and X5;
    Z <= not(Y1 or Y2 or X3);
end behavioural_2;
```

Проектный модуль **Architecture** (синтаксис)

```
architecture имя_архитектуры of имя_объекта is
    декларативная часть
begin
    операторная часть
end [architecture] [имя_архитектуры] ;
```

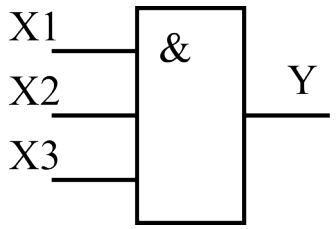
Проектный модуль **Architecture**

Декларативная часть проектного модуля **architecture** определяет объекты, которые могут использоваться в этом проектном модуле. Здесь могут быть определены : типы и подтипы данных ; подпрограммы ; константы ; атрибуты ; сигналы ; компоненты и т. д.

Операторная часть проектного модуля **architecture** определяет структуру и/или выполняемую функцию объекта, определенного проектным модулем **entity**, к которому он относится.

Операторная часть проектного модуля **architecture** может содержать *только параллельные* операторы.

Развёрнутый пример объявления проектного модуля **Architecture**




```
entity and_logic is
   port (
      X1,X2,X3 : in std_logic;
      Y        : out std_logic
   );
end and_logic;
```

Имя архитектуры *Имя объекта*

Декларативная часть

Операторная часть

```
architecture behavioural of and_logic is
   signal Xi : std_logic;
begin
   Xi <= X1 and X2;
   Y  <= Xi and X3
end behavioural;
```

Проектный модуль **Configuration**

Проектный модуль **Configuration** предназначен для сопоставления объектов проекта – компонентов и архитектур. Он выполняет следующие функции:

- позволяет задать объект, соответствующий конкретному компоненту проекта;
- позволяет задать конкретную архитектуру объекту, у которого их несколько.

Проектный модуль **Configuration** (синтаксис)

```
configuration имя_конфигурации of имя_объекта is
    [декларативная часть]
    блок конфигурации
end [architecture] [имя_архитектуры] ;
```

Проектный модуль **Configuration** (декларативная часть)

Декларативная часть проектного модуля **Configuration** может содержать следующие элементы:

- оператор выбора элементов в область видимости проектного модуля **use**;
- оператор определения атрибута (**attribute**);
- оператор определения группы (**group**).

Проектный модуль **Configuration** (блок конфигурации)

```
for идентификатор
    [оператор use]
    элемент конфигурации
end for ;
```

В качестве идентификатора могут использоваться:

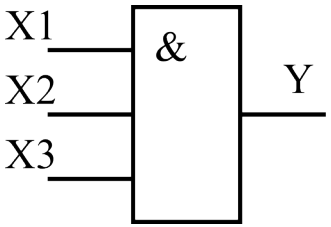
- имя архитектуры;
- метка оператора **block**;
- метка оператора **generate**.

В качестве элемента конфигурации могут использоваться:

- конфигурация блока;
- конфигурация компонента.

Проектный модуль **Configuration** (пример)

Простейшая форма конфигурации – *конфигурация по-умолчанию*



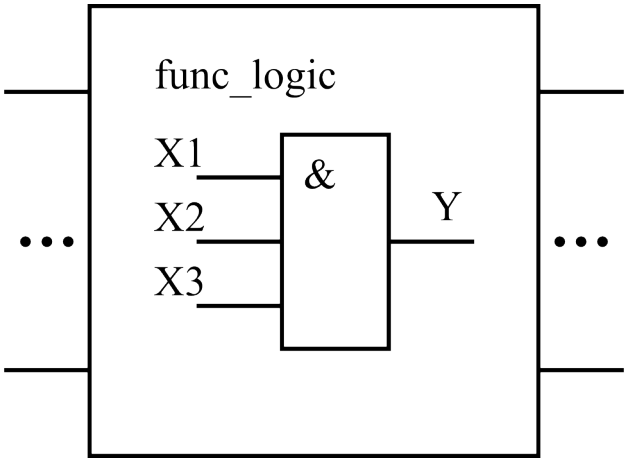
Имя конфигурации *Имя объекта*

```
configuration config1 of and_logic is
   for and_arch1 }  Имя архитектуры
   end for;
end config1;

configuration config2 of and_logic is
   for and_arch2
   end for;
end config2;
```

Проектный модуль **Configuration** (пример)

Конфигурация компонентов



Имя конфигурации *Имя объекта*

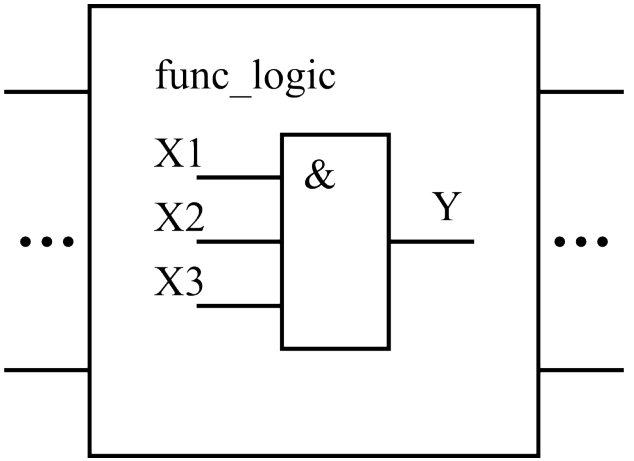
```
configuration config1 of func_logic is
    for func_arch                          } Имя архитектуры
        for and_inst:and_logic use configuration work.config1;
        end for;
    end for;
end config1;
```

Имя установленного компонента

Имя объекта компонента

Проектный модуль **Configuration** (пример)

Конфигурация компонентов с использованием пары **entity – architecture**



Имя конфигурации *Имя объекта*

```
configuration config1 of func_logic is
    for func_arch                          } Имя архитектуры
        for and_inst:and_logic use entity work.and_logic (and_arch1);
        end for;
    end for;
end config1;
```

Имя установленного компонента

Имя объекта компонента *Имя архитектуры компонента*