Модуль

Основным архитектурным элементом в HDL языках является модуль. Модуль HDL языка похож на объект или экземпляр класса из ООП, но имеет некоторые отличия. Функций и методов у модуля нет, но есть процессы. Вызывать их проихвольно нельзя, так как все процессы модуля выполняются всё время, как бы в цикле, без передышки. Со внешним миром модуль общается при помощи портов. Порты похожи на свойства (Property) из ООП. Так же как и свойства порты бывают трёх видов:

- Входные порты. Используются для ввода данных в модуль. Похожи на свойства, имеющие только сеттер. Входное свойство изнутри модуля можно только читать, а снаружи только писать. В VHDL обозначаются ключевым словом **in**, а в Verilog словом **input**.
- Выходные порты. Используются для вывода данных из модуля. Похожи на свойства, имеющие только геттер. Выходное свойство изнутри модуля можно только писать, а снаружи только читать. В VHDL обозначаются ключевым словом **out**, а в Verilog словом **output**.
- Двунаправленные порты. Похожи на свойства имеющие как геттер, так и сеттер. Соответственно их можно читать и писать с любой стороны. Так же как и двунаправленные свойства двунаправленные порты таят в себе одну опасность, а именно: «Если Вы что-то записали в двунаправленный порт, то совершенно не факт, что вы прочтёте из него то же самое». Так происходит по причине того, что двунаправленный порт не хранит данные сам, а является лишь эквивалентом пары противоположно направленных портов, имеющих одно имя. То есть Когда Вы пишите в двунаправленный порт, то пишите в один порт, а когда читаете, то читаете уже из другого.

В языках Verilog и SystemVerilog пустой модуль объявляется так:

```
module ModuleName();

endmodule
```

А это пример модуля с кодом:

```
module ModuleName#(
    integer TEMPLATE_PARAMETER_NAME_A,
    integer TEMPLATE_PARAMETER_NAME_B
)(
    input clock,
    input reset,
    input rx,
    output reg[7:0] data,
    output rcv
);
    // Тут, собственно, и будет весь код.
    // Так выглядит объявление линии, т.е. асинхронного сигнала.
    wire signal_name;
    // Так выглядит объявление многоразрядной линии.
    // (точнее 17 разрядной)
    wire [16:0] signal_name;
    // Так объявляется одноразрядный регистр
    reg  register_name;
    // А так двадцати-разрядный регистр
    reg [19:0]     register_name;
    // А такой тип есть только в SystemVerilog.
    // Он может быть как сигналом, так и регистром.
    logic          logic_name;
```

```
     // Пример асинхронного присвоения вне процесса.
     assign rcv = logic_name & register_name;

     // Так выглядит процесс, синхронизированный по сигналу clock.
     always@(posedge clock) begin
               // В каждом синхронном процессе должен быть сброс.
               if (reset) begin
                    // Он позволяет установить начальные значения сигналам
                    data <= 0;
               end else begin
                    data <= {data[6:0], rx};
               end
     end
endmodule
```

В языке VHDL объявление модуля делится на две части.

Интерфейс

```
entity ModuleName is
     generic()
     port();
end entity ModuleName;
```

Архитектура.

```
architecture ArchitectureName of ModuleName is
begin

end architecture ArchitectureName;
```

Тот же пример на VHDL:
Сначала объявляем интерфейс:

```
-- Такие вот в VHDL коментарии.

-- А так подключаются библиотеки
library IEEE;
use IEEE.std_logic_1164.all;

-- Это объявление интерфейса модуля
entity ModuleName is
     generic(
               TEMPLATE_PARAMETER_NAME_A: integer;
               TEMPLATE_PARAMETER_NAME_B: integer
     )
     port(
               clock: in std_logic;
               reset: in std_logic;
               rx: in std_logic;
               data: out std_logic_vector(7 downto 0);
               rcv: out std_logic
          );
end entity ModuleName;
```

Потом архитектуру:

```
architecture ArchitectureName of ModuleName is
     -- Все сигналы в VHDL объявляются отдельно, как в Pascal.
     -- Так выглядит объявление линии,
     --  как синхронной, так и асинхронной.
     --  В VHDL нельзя указать ограничить синхронность линии.
     signal signal_name: std_logic;
     -- Так выглядит объявление многоразрядной линии.
     -- (точнее 17 разрядной)
     signal signal_name: std_logic_vector(16 downto 0);
     -- Так объявляется одноразрядный регистр
     -- Собственно, так же, как и любые другие линии.
     signal register_name: std_logic;
     -- А так двадцати-разрядный
     signal register_name: std_logic_vector(19 downto 0);
     -- Просто ещё один сигнал.
     signal logic_name: std_logic;
begin
     -- Пример асинхронного присвоения вне процесса.
     rcv <= logic_name and register_name;

     -- Так выглядит процесс, синхронизированный по сигналу clock.
     PROCESS_NAME: process(clock)
     begin
               if (clock'event and clock = '1') then
                    -- В каждом синхронном процессе должен быть сброс.
                    if (reset = '1') then
                         -- Он позволяет установить начальные значения
                         data <= 0;
                    else
                         data <= data(6 downto 0)&rx;
                    end if;
               end;
     end process;
end architecture ArchitectureName;
```

Разберём примеры более детально

Объявления.

Шаблонные параметры и порты

Модули в HDL языках поддерживают шаблонные параметры. Работают эти шаблонные параметры так же, как и шаблонные параметры в C++. С их помощью можно, например, создать модуль оперативной памяти с параметризуемым размером.

Основным же средством общения с внешним миром для модуля являются порты. Порты бывают, как объяснялось ранее, бывают: входные, выходные и двунаправленные. Изнутри модуля порты ведут себя как обычные переменные, т.е. их можно читать и писать по тем же правилам.

В Verilog

```
// Это объявление интерфейса модуля
module ModuleName#(
     // Это шаблонные параметры
    integer TEMPLATE_PARAMETER_NAME_A,
    integer TEMPLATE_PARAMETER_NAME_B
)(
     // А это порты
     // Входные
    input clock,
    input reset,
    input rx,
    // И Выходные
    output reg[7:0] data,
    output rcv
);
```

В VHDL

```
-- Это объявление интерфейса модуля
entity ModuleName is
     -- Это шаблонные параметры
     generic(
               TEMPLATE_PARAMETER_NAME_A: integer;
               TEMPLATE_PARAMETER_NAME_B: integer
     )
     -- А это порты
     port(
               -- Входные
               clock: in std_logic;
               reset: in std_logic;
               rx: in std_logic;
               -- И Выходные
               data: out std_logic_vector(7 downto 0);
               rcv: out std_logic
          );
end entity ModuleName;
```

Переменные (Сигналы, Линии, Регистры)

В Verilog

В Verilog разрядность линии указывается так: **[***НомерСтаршегоРазряда***:***НомерМладшегоРазряда***]**
Так же можно выделить из линии часть разрядов.

```
     // Так выглядит объявление линии, т.е. асинхронного сигнала.
     wire signal_name;
     // Так выглядит объявление многоразрядной линии.
     // (точнее 17 разрядной)
     wire [16:0] signal_name;
     // Так объявляется одноразрядный регистр
     reg  register_name;
```

```
    // А так двадцати-разрядный регистр
    reg [19:0]     register_name;
    // А такой тип есть только в SystemVerilog.
    // Он может быть как сигналом, так и регистром.
    logic          logic_name;
```

В VHDL

В VHDL разрядность линии указывается так:

(*НомерСтаршегоРазряда* **downto** *НомерМладшегоРазряда***)**

Так же можно выделить из линии часть разрядов.

```
    -- Так выглядит объявление линии,
    --  как синхронной, так и асинхронной.
    --  В VHDL нельзя указать ограничить синхронность линии.
    signal signal_name: std_logic;
    -- Так выглядит объявление многоразрядной линии.
    -- (точнее 17 разрядной)
    signal signal_name: std_logic_vector(16 downto 0);
    -- Так объявляется одноразрядный регистр
    -- Собственно, так же, как и любые другие линии.
    signal register_name: std_logic;
    -- А так двадцати-разрядный
    signal register_name: std_logic_vector(19 downto 0);
    -- Просто ещё один сигнал.
    signal logic_name: std_logic;
```

Асинхронное присвоение

Асинхронное присвоение фактически создаёт именованный синоним выражению, стоящему справа от знака равенства. То есть выражение слева от знака равенства будет всегда (Всё время) равно выражению справа от знака равенства. Задержка (в наносекундах) может возникнуть только на выполнение операции, если выражение справа не является простой переменной или константой.

В Verilog

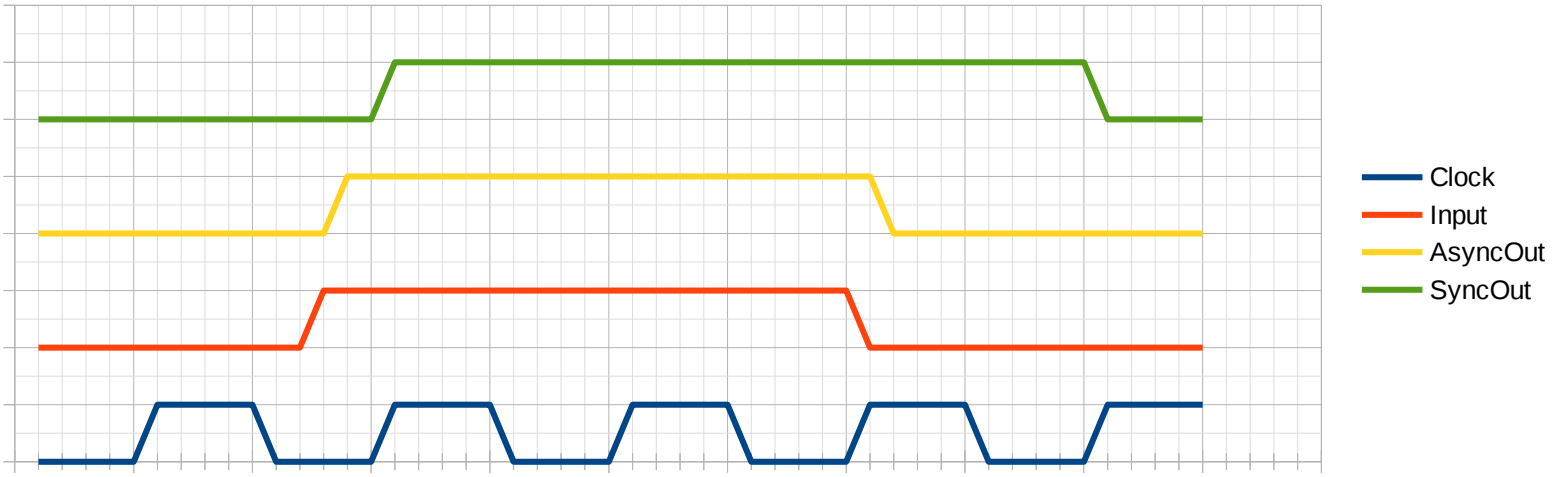
```
    // Пример асинхронного присвоения вне процесса.
    assign rcv = logic_name & register_name;
```

В VHDL

```
    -- Пример асинхронного присвоения вне процесса.
    rcv <= logic_name and register_name;
```

Синхронный процесс

Суть синхронного процесса в том, что он происходит не всё время, а только по фронту тактового сигнала, который периодически устанавливается то в 0 то в 1. Ниже на графике видно, что асинхронный повторитель изменяет своё значение с почти сразу же после того, как изменился вход. Синхронный же выход выбирает своё значение лишь тогда, когда тактовый сигнал меняется с 0 на 1. Между такими моментами синхронный сигнал запоминает своё состояние. В реальных электрических схемах есть лишь один синхронный логический элемент — это триггер, так же известный как одноразрядный регистр.



Кроме того в каждом синхронном процессе должен быть сброс. Это сигнал, при выставлении которого в 1 все регистры, которые присваиваются в данном синхронном процессе, должны быть установлены в состояние по умолчанию. Наличие сигнала сброса позволяет гарантировать, что после включения электрической схемы все её регистры будут иметь некоторое предсказуемое состояние, устанавливаемое им при сбросе.

В Verilog

```
// Так выглядит процесс, синхронизированный по сигналу clock.
always@(posedge clock) begin
          // В каждом синхронном процессе должен быть сброс.
          if (reset) begin
               // Он позволяет установить начальные значения сигналам
               data <= 0;
          end else begin
               data <= {data[6:0], rx};
          end
end
```

В VHDL

```
-- Так выглядит процесс, синхронизированный по сигналу clock.
PROCESS_NAME: process(clock)
begin
          if (clock'event and clock = '1') then
               -- В каждом синхронном процессе должен быть сброс.
               if (reset = '1') then
                    -- Он позволяет установить начальные значения
                    data <= 0;
               else
                    data <= data(6 downto 0)&rx;
               end if;
          end;
end process;
```

Логические операции

В Verilog

Оператор	Назначение	Пример
{}	Конкатенация	assign c = {a, 0, 1, b}
!	Логическое отрицание	assign c = !a
&&	Логическое И	assign c = a && b
||	Логическое или	assign c = a || b
==	Равенство	assign c = (a==b)
!=	Неравенство	assign c = (a!=b)
~	Побитовая инверсия	assign c = ~a
&	Побитовое И	assign c = a&b
|	Побитовое ИЛИ	assign c = a|b
^	Побитовое исключающее ИЛИ	assign c = a^b
<<	Сдвиг влево	assign c = a << b
>>	Сдвиг вправо	assign c = a >> b
?:	Условное присвоение	assign c = a?b:d

В VHDL

Оператор	Назначение	Пример
&	Конкатенация	c <= a&0&1&b
=	Равенство	if (a=b) then end if
/=	Неравенство	if (a/=b) then end if
not	Побитовая инверсия	c <= not a
and	Побитовое И	c <= a and b
or	Побитовое ИЛИ	c <= a or b
xor	Побитовое исключающее ИЛИ	c <= a xor b
shl	Сдвиг влево	c <= a shl b
shr	Сдвиг вправо	c <= a shr b

Арифметические операции

В Verilog

Оператор	Назначение	Пример
+	Сложение	`assign c = a + b`
-	Вычитание	`assign c = a - b`
*	Умножение	`assign c = a * b`
/	Деление	`assign c = a / b`

В VHDL

В VHDL с арифметикой дела обстоят сложно. Складывать битовые вектора, словно целые числа, в VHDL невозможно. Для того, что бы работать с арифметикой необходимо подключать библиотеки.

Для работы с беззнаковой арифметикой:

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

Для работы со знаковой арифметикой:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
```

Арифметические операции поддерживаются те же, носинтаксис, с учётом преобразований типов, несколько отличается.

Во всех четырёх примерах далее считаем, что сигналы a, b, c имеют разрядность 32. С другой разрядностью это тоже будет работать, только вместо 32 надо будет подставить другое число.

Беззнаковая арифметика

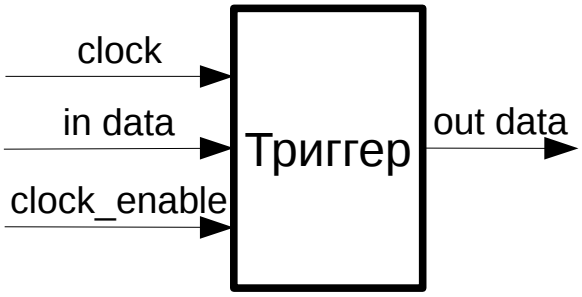
	Пример
+	`a <= conv_std_logic_vector(unsigned(b) + unsigned(c), 32);`
-	`a <= conv_std_logic_vector(unsigned(b) - unsigned(c), 32);`
*	`a <= conv_std_logic_vector(unsigned(b) * unsigned(c), 32);`
/	`a <= conv_std_logic_vector(unsigned(b) / unsigned(c), 32);`

Знаковая арифметика

	Пример
+	`a <= conv_std_logic_vector(signed(b) + signed(c), 32);`
-	`a <= conv_std_logic_vector(signed(b) - signed(c), 32);`
*	`a <= conv_std_logic_vector(signed(b) * signed(c), 32);`
/	`a <= conv_std_logic_vector(signed(b) / signed(c), 32);`

Вкл/Выкл тактового сигнала (Clock enable). Управляемая память.

Несмотря на то, что обычный синхронный процесс позволяет сохранять информацию в переменной от одного фронта тактового сигнала до другого, его будет недостаточно для того, что бы запоминать информацию на значительное время. Для этих целей используется механизм, называемый «Clock enable», позволяющий управлять реакцией регистра на фронт тактового сигнала. Такой механизм уже заложен во все триггеры и регистры в ПЛИС и для его реализации в HDL языках существует определённый шаблон, в соответствии с которым нужно писать код.



В Verilog

```
// Собственно обычный синхронный процесс.
always@(posedge clock) begin
          // Так же как и всегда делаем сброс.
          if (reset) begin
               data <= 0;
          end else begin
               // Присваиваем не просто так, а по условию.
               if (clock_enable) out_data <= in_data;
          end
end
```

В VHDL

```
-- Собственно обычный синхронный процесс.
PROCESS_NAME: process(clock)
begin
          if (clock'event and clock = '1') then
               -- Так же как и всегда делаем сброс.
               if (reset = '1') then
                    data <= 0;
               else
                    -- Присваиваем не просто так, а по условию.
                    if (clock_enable = '1') then
                         out_data <= in_data;
                    end if;
               end if;
          end;
end process;
```

Обратите внимание на то, что условие включения/выключения тактового сигнала, для обоих языков, находится внутри условия, реализующего сброс, и, именно, в блоке **else**, поскольку каждый clock_enable включатет или выключает реакцию на тактовый сигнал только для соответствующего регистра (*т.е. соответствующей переменной*), а не для всего процесса в целом. Располагать их нужно

именно так и никак иначе. Если это необходимо, допускается устанавливать несколько условий включения/выключения тактового сигнала.

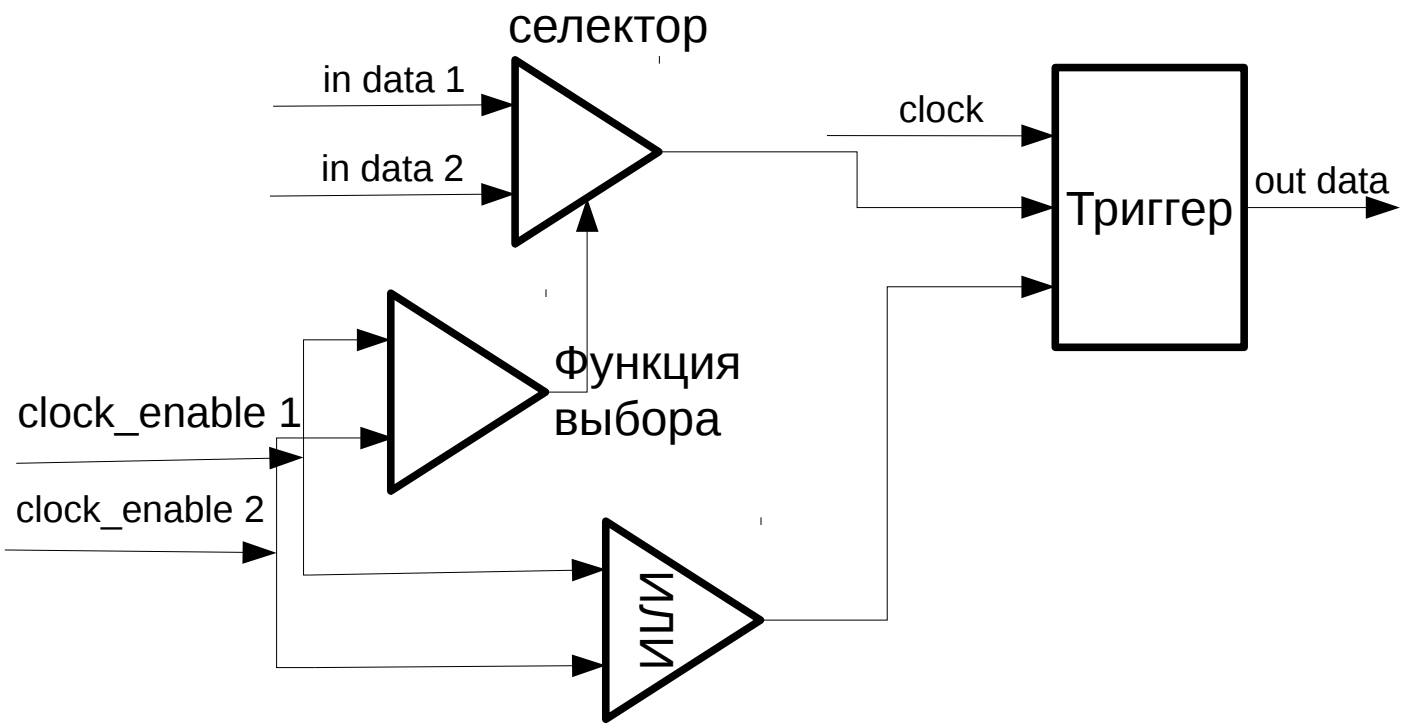
В Verilog

```
    // Собственно обычный синхронный процесс.
    always@(posedge clock) begin
              // Так же как и всегда делаем сброс.
              if (reset) begin
                   data <= 0;
              end else begin
                   // Присваиваем не просто так, а по условию.
                   if (clock_enable_1) out_data <= in_data_1;
                   else if (clock_enable_2) out_data <= in_data_2;
              end
    end
```

В VHDL

```
    -- Собственно обычный синхронный процесс.
    PROCESS_NAME: process(clock)
    begin
              if (clock'event and clock = '1') then
                   -- Так же как и всегда делаем сброс.
                   if (reset = '1') then
                        data <= 0;
                   else
                        -- Присваиваем не просто так, а по условию.
                        if (clock_enable_1 = '1') then
                             out_data <= in_data_1;
                        elsif (clock_enable_2 = '1') then
                             out_data <= in_data_2;
                        end if;
                   end if;
              end;
    end process;
```

В таком случае, в результате, будет синтезироваться такая схема:



При этом функция выбора может получиться различной, в зависимости от порядка следования условий и будет, фактически, задавать их приоритет.

Некоторые НЕ СИНТЕЗИРУЕМЫЕ конструкции.

Внимание: Использовать только при симуляции!

Генерация сигналов

В Verilog

```
    // Генерируем тактовую частоту с периодом 20.
    // 20, потому что 20 = 2 * 10 - то время,
    // чрез которое тактовый сигнал вернётся в то же состояние.
    initial begin // initial Выполняется лишь раз, при старте, и всё.
        Clock <= 0;
        forever begin // Позволяет выполнять операции в вечном цикле
            #10;// Задержка на 10 ns. (forever без задержки выдаст ошибку)
            Clock <= !Clock;
        end
    end

    // Генерируем сигнал сброса.
    initial begin // И этот тоже initial.
        Reset <= 1;
        @(posedge Clock) // Ждём ближайший фронт нашего тактового сигнала
        Reset <= 0;
    end

    // Собственно генерируем данные для теста
    int i, j;
    initial begin // Все initial выполняются одновременно, при старте
        Btns <= 0;
        // Цикл, генерирующий данные для симуляции
        for (i = 0; i < 4; i += 1) begin
            // Ничего не делающий цикл, пропускающий 8 тактов
            for (j = 0; j < 8; j += 1) @(posedge Clock);
            // А тут какие-то операции
            Btns += 1;
        end
    end
```

В VHDL

```
    -- Генерируем тактовую частоту с периодом 20 ns.
    -- 20 ns, потому что 20 ns = 2 * 10 ns - то время,
    -- чрез которое тактовый сигнал вернётся в то же состояние.
    process begin
        Clock <= '0';
        loop
            wait for 10 ns;
            Clock <= not Clock;
        end loop;
    end process;

    -- Генерируем сигнал сброса.
    process begin
        Reset <= '1';
        wait until rising_edge(Clock);
        Reset <= '0';
        wait;
    end process;

    -- Собственно генерируем данные для теста
    process
    variable i, j: integer;
    begin
        Btns <= x"0";
        for i in 0 to 3 loop
            for j in 0 to 7 loop
                wait until rising_edge(Clock);
            end loop;
            Btns <= conv_std_logic_vector(unsigned(Btns) + 1, 4);
        end loop;
        wait;
    end process;
```

Некоторые СИНТЕЗИРУЕМЫЕ конструкции

Которые так же будут полезны при симуляции

Использование другого модуля

В Verilog

```
-- Синтаксис такой:
-- <Имя Вашего Модуля>  <Имя очередного экземпляра модуля>
    SampleVerilog_2     SampleVerilog_2_Item (
-- .<Имя Порта модуля> (    <Имя сигнала>  )
        .Clock          (       Clock       ),
        .Reset          (       Reset       ),
        .Btns           (       Btns        ),
        .Leds           (       Leds        )
    );
-- Ставить так же много пробелов, как у меня тут, не обязательно.
-- Это просто для наглядности.
```

В VHDL

```
-- Синтаксис такой:
-- <Имя очередного экземпляра модуля>: <Имя Вашего Модуля>
    SampleVHDL_2_Item: SampleVHDL_2
    port map  (
-- .<Имя Порта модуля> =>    <Имя сигнала>
        Clock           =>    Clock,
        Reset           =>    Reset,
        Btns            =>    Btns,
        Leds            =>    Leds
    );
-- Ставить так же много пробелов, как у меня тут, не обязательно.
-- Это просто для наглядности.
```