

# VHDL

- Проектирование больших вычислительных устройств (ВУ)
- Проект на VHDL -объединение структуры ВУ и алгоритма его функционирования
- Проект на VHDL – самодокументированный
- Высокая надежность проекта
- Проект на VHDL - универсальный проект
- Проект на VHDL - портативный проект
- Проект на VHDL - долгоживущий проект

# Типы данных

- **Перечисляемый тип** определяется как список (перечисление) всех возможных значений данного типа. Объявления этого типа выглядят как:

```
type \имя типа\ is (\перечисляемый литерал\ {,\ перечисляемый литерал\});
```

- **Целый тип.** Объявление этого типа выглядит как:

```
type \имя типа\ is range \диапазон целых\;
```

- **Регулярный тип** представляет собой множество элементов одинакового типа. Различают неограниченные и ограниченные регулярные типы.

Неограниченный тип объявляется как:

```
type \имя регулярного типа\ is array (\имя типа диапазона\range<>) of \имя  
типа элемента\; где \имя типа диапазона\ - имя типа integer или какого-либо  
подтипа от integer.
```

Ограниченный регулярный тип объявляется как:

```
type \имя регулярного типа\ is array (\диапазон целых\ of \имя типа  
элемента\);
```

# Предопределенные типы данных

- **type** boolean is (false, true);
- **type** bit is ('0', '1');
- **type** integer is range -2147483647 to 2147483647;
- **subtype** natural is integer range 0 to 2147483647;
- **type** bit\_vector is array (natural range <>) of bit;

# Тип STD\_LOGIC

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

```
type std_ulogic is ( 'U', -- Uninitialized  
                    'X', -- Forcing Unknown  
                    '0', -- Forcing 0  
                    '1', -- Forcing 1  
                    'Z', -- High Impedance  
                    'W', -- Weak Unknown  
                    'L', -- Weak 0  
                    'H', -- Weak 1  
                    '-' -- Don't care  
                    );
```

```
type std_logic_vector is array ( natural range <> ) of std_logic;
```

# Объекты языка VHDL

## Сигнал.

- Сигналом является объект, который переносит значение от одного процесса к другому и вместе с ним - синхронизирующее воздействие. Сигнал может быть запомнен в своей истории и воспроизведен в симуляторе в виде графика или таблицы. Объявление сигнала выглядит как:

```
signal : \идентификатор\{,\идентификатор\} :  
\тип\:=[\начальное значение\];
```

где - \начальное значение\ - выражение, представляющее константу, значение которой принимает сигнал перед первым запуском процесса.

```
signal cnt: natural range 0 to 127:=0;
```

# Объекты языка VHDL

**Константой** является объект, не изменяющий свое значение при вычислениях. После объявления константы присваивание ей значения запрещено (кроме случая отложенной константы).

Пример объявления константы:

```
constant thousand: integer:=1000;
```

# Объекты языка VHDL

## Переменная.

- Переменной является объект, хранящий значение в пределах операторов процесса, функции или процедуры. В отличие от сигнала, присваивание переменной выполняется немедленно.

Упрощенный синтаксис объявления переменной:

`\объявление переменной\::=`

`variable идентификатор\ {,\идентификатор\} :\тип\`

`[:=\начальное значение\];`

- Пример объявления переменной:

`variable tmp: integer range -128 to 127:=0;`

# Объекты языка VHDL

## Порт.

- В структуре программы VHDL выделяются объекты проекта, называемые entity. Не путать объекты языка с объектом проекта. Это связано с тем, что, во-первых, object и entity переводятся одинаково, во-вторых, в Language Reference Manual сигналы, переменные и др. в некоторых декларативных местах обзываются как entities.
- Порт представляет собой интерфейсный сигнал объекта проекта. Как и в декларации сигнала, в декларации порта указывается его идентификатор, тип, начальное значение. Дополнительно указывается режим работы: in - прием, out - передача, inout - прием и передача. Упрощенный синтаксис объявления портов объекта проекта следующий:

```
\объявление портов\ ::= port (\объявление порта\ {; \объявление  
порта\}); \объявление порта\ ::= \идентификатор\ : in | out | inout \тип\  
[:=\начальное значение\]
```



# Операции

Тип операции

Символ или ключевое слово

Логические  
Сравнения  
Сдвига  
Сложения  
Унарные (знак)  
Умножения  
Различные

**and, or, nand, nor, xor, xnor**  
**=, /=, <, <=, >, >=**  
**sll, srl, sla, sra, rol, ror**  
**+, -, &** (конкатенация)  
**+, -**  
**\*, /, mod, rem**  
**\*\* , abs, not**

# Структура программы на VHDL

Дискретная система может быть представлена в VHDL как объект проекта. Объект – это основная составная часть проекта. Объект может быть использован в другом проекте, который, в свою очередь, описан как объект или может являться объектом более высокого уровня в данном проекте. Не путать объект проекта с объектами языка.

Объект проекта описывается набором составных частей проекта, таких как: объявление объекта называемое **entity**, тело архитектуры объекта (или просто архитектура), именуемое **architecture**. Каждая из составных частей объекта может быть скомпилирована отдельно. Составные части проекта сохраняются в одном или нескольких текстовых файлах с расширением .VHD. В одном файле может сохраняться несколько объектов проекта.

Объект проекта обычно описывается согласно синтаксису:

```
\объект проекта\ ::= [\описание library\  
                    [\описание use\  
                    \объявление объекта\  
                    \тело архитектуры\  
                    ]]
```

# Объявление объекта

Объявление объекта указывает, как объект проекта выглядит снаружи и каким образом его можно включить в другом объекте в качестве компонента, т.е. он описывает внешний интерфейс объекта. Синтаксис объявления объекта:

```
\объявление объекта\ ::= entity \идентификатор\ is  
    [port (\объявление порта\ {;\объявление  
порта\});]  
    {\объявление в объекте\  
end [entity][\идентификатор\];
```

# Архитектура объекта

Архитектура объекта представляет собой отдельную часть, в которой описано, каким образом реализован объект. Ее синтаксис:

```
\тело архитектуры\ ::= architecture \идентификатор\ of \имя объекта\ is  
    {\объявление в блоке\  
    begin  
        {\параллельный оператор\  
    end [architecture][\идентификатор\];
```

- Идентификатором обозначается имя конкретного тела архитектуры, а имя объекта указывает, который из объектов описан в этом теле. Одному объекту проекта может соответствовать несколько архитектур, в каждой из которых описан один из вариантов реализации объекта.
- Объявление в теле архитектуры такое же, как в блоке и им может быть: объявление и тело процедуры или функции, объявление типа и подтипа, объявление файла, псевдонима, константы, глобальной переменной, объявление и спецификация атрибута, объявление группы, описание **use**, а также объявление компонентов. Объявленные в теле архитектуры типы, сигналы, подпрограммы видимы только в пределах этой архитектуры.
- Исполнительную часть архитектуры составляют параллельные операторы, такие как процесс, блок, параллельное присваивание сигналу и др. Эти операторы исполняются параллельно.
- Так как все операторы в исполнительной части тела архитектуры – параллельные, их взаимный порядок – безразличен. Хорошим стилем считается, когда параллельные операторы ставятся в последовательности, соответствующей цепочкам вершин графа-схемы алгоритма, реализуемого в архитектуре.

# Оператор process

Оператор процесса – это параллельный оператор, представляющий основу языка VHDL. Его упрощенный синтаксис:

```
\оператор процесса\ ::= [postponed] process [(\имя сигнала\ {\, \имя сигнала\})] [is]  
{\объявление в процессе\}  
begin  
{\последовательный оператор\}  
end process;
```

- Объявленными в процессе могут быть: объявление и тело подпрограммы, объявление типа и подтипа, объявление константы, переменной, файла, псевдонима, объявление и спецификация атрибута, объявление группы, описание **use**. То, что объявлено в процессе, имеет область действия (видимость), ограниченную данным процессом.
- Все процессы в программе выполняются параллельно. Процессы обмениваются сигналами, которые выполняют синхронизацию процессов и переносят значения между ними. Если над сигналами определена функция разрешения, то выходы источников сигнала могут объединяться. Сигналы нельзя объявлять в процессах. Процесс невозможно поместить в процесс, так как там есть место только для последовательных операторов.
- В круглых скобках заголовка процесса указывается множество сигналов, по которым процесс запускается – список чувствительности. Это форма оператора процесса, альтернативная процессу с оператором **wait on**, стоящим последним в цепочке последовательных операторов тела процесса. Любой процесс со списком чувствительности может быть преобразован в эквивалентный процесс с оператором **wait on**, стоящим последним в списке последовательных операторов. В операторе процесса со списком чувствительности ставить операторы **wait** не допускается.

# Последовательный оператор присваивания

Последовательные операторы в VHDL вставляются в операторы процессов и исполняются последовательно. Последовательный оператор присваивания выполняет присваивание переменной или сигналу результата выражения. Он выглядит следующим образом:  
`\приемник\ := \выражение\` - для присваивания переменной  
`\приемник\ <= \выражение\` - для присваивания сигнала.

Присваивание переменной отличается от присваивания сигналу. Присваивание переменной выполняется немедленно.

Выполнение оператора присваивания сигналу означает вычисление его выражения и лишь назначение сигналу. Само же присваивание сигналу фактически выполняется в момент остановки процесса по ожиданию события. Поэтому если в одном процессе стоит несколько присваиваний одному сигналу, то истинное присваивание происходит в момент остановки процесса. Если перед остановкой процесса выполнялось чтение этого сигнала (участие его в качестве операнда в выражении) то будет прочитано значение, присвоенное в прошлом запуске процесса.

# Последовательный оператор `wait`

## Оператор ожидания события `wait`.

- На этом операторе выполнение процесса останавливается, в момент остановки выполняются присваивания сигналам и процесс продолжает исполнение при появлении события, которое выбирается этим оператором. Синтаксис оператора `wait`:

```
\оператор wait\ ::= wait [on \имя сигнала\ {\имя сигнала\}]  
    [until \булевское выражение\] [for \выражение времени\];
```

- где ключевое слово `on` начинает список чувствительности, `until` - условие ожидания, а `for` - задержку ожидания. По оператору

`wait on CLK, RST;`

- продолжение выполнения процесса начнется по событию изменения сигналов CLK или RST. По оператору

`wait until CLK='1';`

- продолжение начнется в момент изменения состояния CLK из '0' в '1', т.е. по фронту этого сигнала. Оператор

`wait for CLK_PERIOD;`

- остановит процесс на время, заданное переменной CLK\_PERIOD типа time.
- Возможно комбинирование списка чувствительности, условия ожидания и задержки ожидания в одном операторе. Оператор `wait` без списка чувствительности, условия ожидания и задержки ожидания остановит процесс до конца моделирования.

# Оператор if

Этот условный оператор в зависимости от заданных условий выполняет цепочки последовательных операторов, причем от условия зависит, которая из цепочек операторов выполняется. Упрощенный синтаксис оператора:

```
\оператор if ::= if \условие 1\ then  
{\последовательный оператор 1\  
[ { elif \условие 2\ then  
{\последовательный оператор 2\  
[else  
{\последовательный оператор 3\  
end if;
```

Каждое из условий должно быть выражением, вычисляющим результат булевского типа. При выполнении этого оператора условия проверяются последовательно друг за другом пока результат условия не будет **true**. Тогда выполняется соответствующая этому условию цепочка операторов и выполнение данного оператора **if** прекращается.



# Оператор case

Этот оператор разрешает выполнение одной из цепочек последовательных операторов в зависимости от значения выражения селектора. Его упрощенный синтаксис:

```
\оператор case\ ::= case \простое выражение\ is  
when \альтернативы\ => {\последовательный оператор\  
{when \альтернативы\ => {\последовательный оператор\  
end case ;  
\альтернативы\ := \альтернатива\{ | \альтернатива\}
```

В выражении селектора `\простое выражение\` должен вычисляться целый результат или значение перечисляемого или регулярного типа. Это должно быть простое выражение, а не, например, конкатенация. Каждая из альтернатив `\альтернатива\` должна быть такого же типа, что и `\выражение\` и представлена статическим выражением или диапазоном, например, `0 to 4`. Никакие два значения, получаемые из выражений альтернатив, не должны быть равны друг другу, т.е. множества альтернатив не перекрываются. Последней альтернативой может быть ключевое слово **others**, которое указывает на не перечисленные альтернативы. Если слово **others** не применяется, то в альтернативах должны быть перечислены все возможные значения, принимаемые в селекторе `\выражение\`.

# Оператор параллельного присваивания

Этот оператор имеет такой же синтаксис, как и оператор присваивания сигналу в процессе. Такой оператор эквивалентен оператору процесса, в котором этот оператор повторен в его исполнительной части, а последним оператором стоит оператор **wait** со списком чувствительности. Например, следующие два оператора эквивалентны:

```
ADDER:A<=B+C;
```

```
ADDER_P:process begin
```

```
    A<=B+C;
```

```
    wait on B,C;
```

```
end process;
```

# Оператор условного параллельного присваивания

Оператор условного параллельного присваивания имеет синтаксис:

```
\условное параллельное присваивание\ ::= \имя\<= [\способ задержки\  
    {\график\ when \булевское выражение\ else }  
    \график\[when \булевское выражение\];
```

где определение способа задержки и графика представлено выше при описании оператора присваивания сигналу.

Любой оператор условного параллельного присваивания имеет эквивалентное представление в виде процесса:

```
cntrl<= one when st=1 else  
    two when st=2 or st=3 else  
    three;
```

эквивалентен оператору

```
process(st,one,two,three)
```

```
begin
```

```
  if st=1 then
```

```
    cntrl<= one;
```

```
  elsif st=2 or st=3 then
```

```
    cntrl<= two;
```

```
  else
```

```
    cntrl<=three;
```

```
  end if;
```

```
end process;
```

# Оператор селективного параллельного присваивания

Оператор селективного параллельного присваивания имеет синтаксис:

```
\селективное параллельное присваивание\ ::= with \выражение\ select  
  {\имя\<= [\способ задержки\]{\график\ when \альтернативы\},  
   \график\[\bwhen others ]};
```

где \альтернативы\ имеют то же значение, что и в операторе `case`. Этот оператор эквивалентен соответствующему процессу, как например, оператор:

```
with st select  
  cntrl<= one when 1,  
    two when 2 to 3,  
    three when others;
```

выполняет такие же действия, что и процесс:

```
process(st,one,two,three)  
begin  
  case st is  
    when 1 => cntrl<= one;  
    when 2 to 3 => cntrl<= two;  
    when others => cntrl<= three;  
  end case;  
end process;
```

Требования к выражению селектора и альтернативам оператора такие же, как у оператора `case`. Так, каждая из альтернатив должна быть такого же типа, что и \выражение\ и представлена статическим выражением или диапазоном. Никакие два значения, получаемые из выражений альтернатив, не должны быть равны друг другу.

# Оператор вставки компонента

Оператор вставки компонента - играет основную роль для реализации иерархического проектирования. Его синтаксис:  
`\оператор вставки компонента\ ::= \метка экземпляра элемента\ :`

`\вставляемый элемент\`

`[generic map(\связывание настроечной константы\  
{, \связывание настроечной константы\});]`

`[port map (\связывание порта\  
{, \связывание порта\});]`

`\вставляемый элемент\ ::= [component] \имя компонента\`