

Московский Государственный Университет Приборостроения и Информатики

Основы языка VHDL



Цели и задачи

- Получение навыков создания моделей при помощи языка VHDL
- Описание цифровых систем при помощи логических функций
- Иерархическое описание цифровых систем средствами языка VHDL

Структура курса

- Введение
- Основные элементы языка VHDL
- Методы моделирования на VHDL
- Синтез цифровых систем
- Иерархический дизайн цифровых систем

Основы языка VHDL

Введение



Основы VHDL

- Язык VHDL утвержден на уровне отраслевых стандартов (IEEE) как программное средство для описания аппаратных систем.
- Язык высокого уровня применимый как для моделирования так и для синтеза.

Основные определения

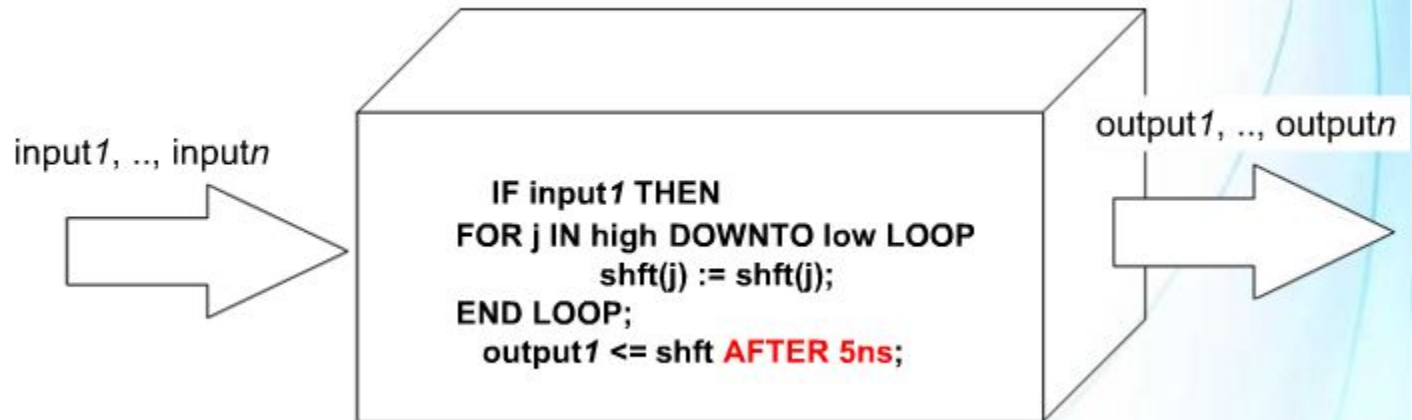
- HDL – Hardware description language, язык описания аппаратных средств
- Поведенческое моделирование: (Behavior modeling) – описание компонента на основе вход-выходных зависимостей
- Структурное моделирование (Structural modeling): описание системы на уровне компонентов и связей между ними

Основные определения (продолжение)

- Register Transfer Layer (RTL) – Уровень проектирования (Уровень регистрового обмена)
- Синтез – трансляция с уровня описания на языке HDL на уровень электрических цепей
- Процесс – исполняемая единица в языке VHDL

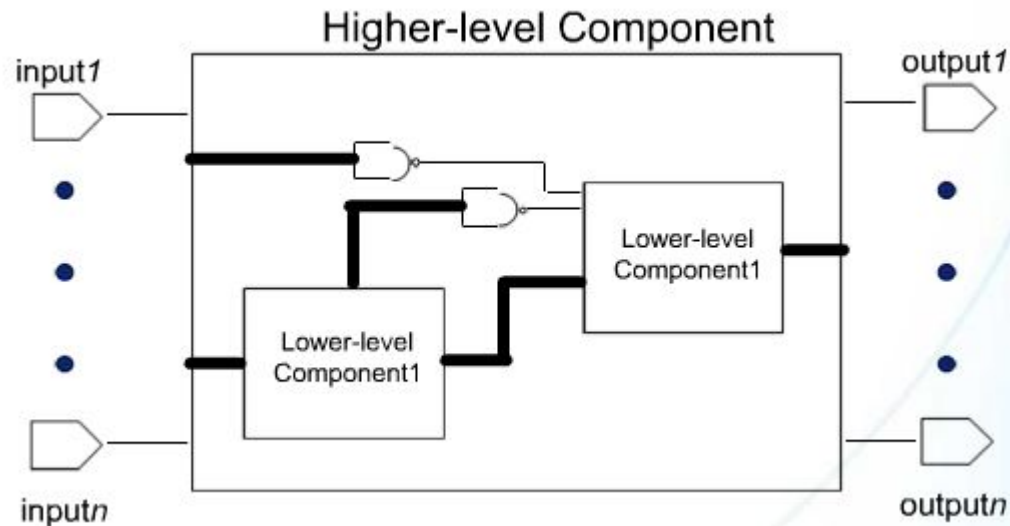
Поведенческое моделирование

- Используется описание поведения элемента
- Не используется информация о связях между компонентами

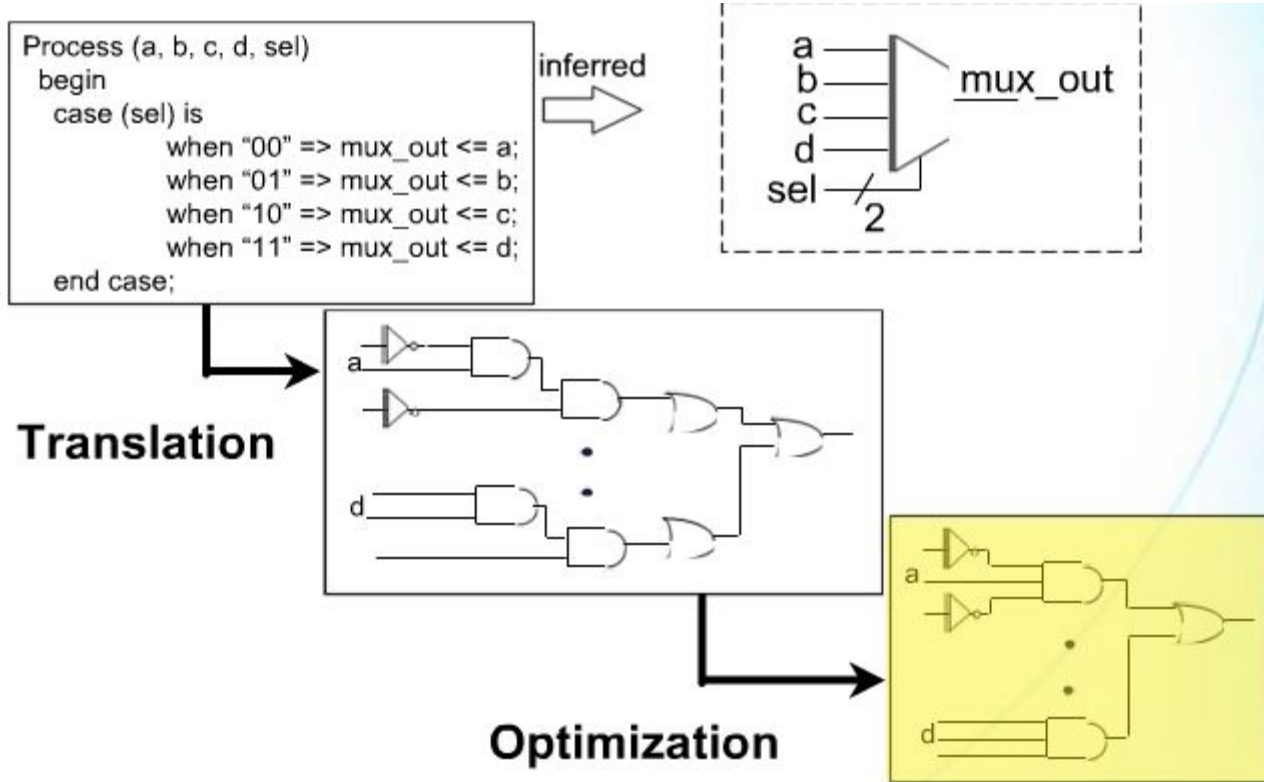


Структурное моделирование

- Функциональное и структурное описание объекта моделирования
- Возможно использование аппаратно зависимых ресурсов



RTL Синтез



VHDL в сравнении с другими HDL языками

■ VHDL:

- «Создать устройство выход которого изменяется только в момент когда возникает переход с низкого на высокий уровень на определенном входе. В этот момент выход должен получить значение сигнала на входе»
- Результат: процедура синтеза создаст триггер с синхронизацией «по фронту».

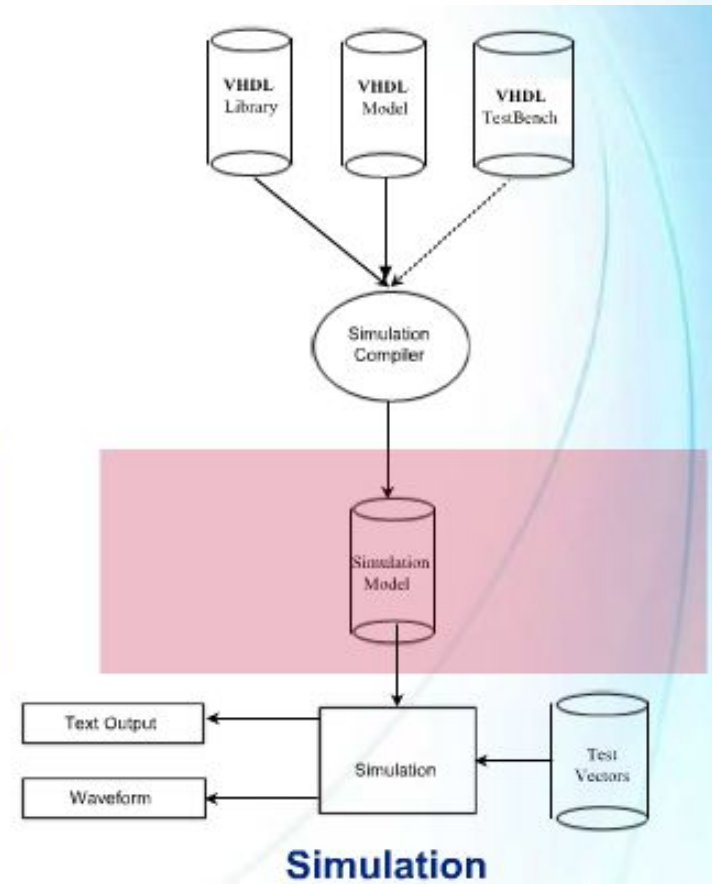
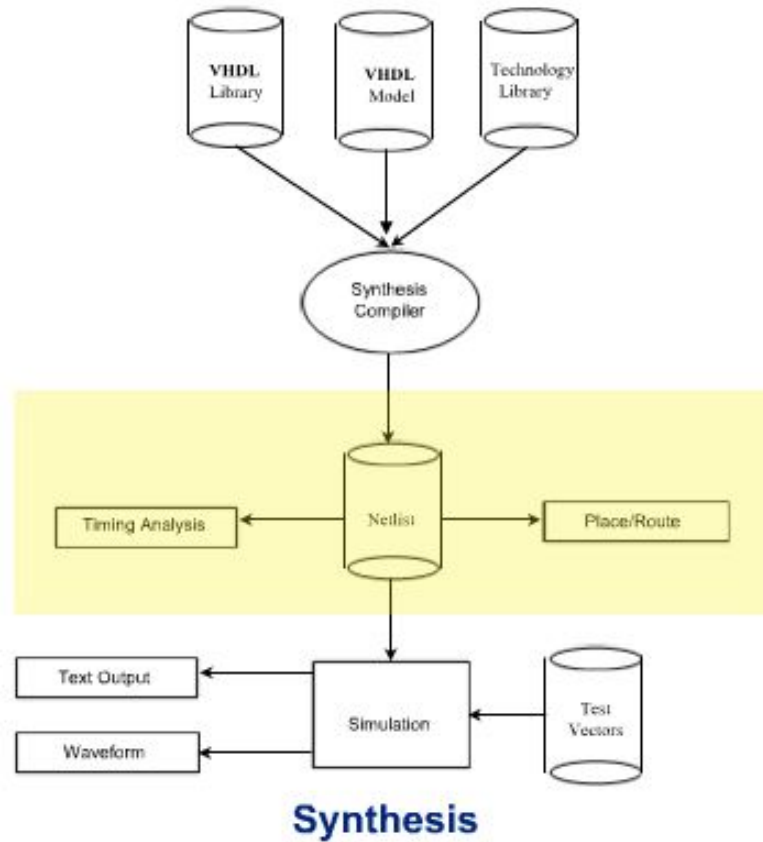
■ ABEL, PALASM, AHDL:

- «Создать D-триггер»
- Результат: в зависимости от средств синтеза будет создан D-триггер с синхронизацией по фронту или по срезу.

Некоторые особенности VHDL

- Два подмножества конструкций языка:
 - Синтезируемые конструкции
 - Конструкции для моделирования
- Язык VHDL нечувствителен к регистру.
- Операторы языка VHDL разделяются символом ;
- Нечувствителен к количеству пробелов
- Комментарии в VHDL начинаются с символа - -

Типовые последовательности VHDL



Основы языка VHDL

Основные конструкции VHDL



Основные элементы языка VHDL

■ Entity

- Используется для описания интерфейса модели.

■ Architecture

- Используется для описания поведения модели.

■ Configuration

- Используется для связывания объектов Entity и Architecture.

■ Package

- Набор конструкций, которые могут быть использованы другими VHDL модулями (аналог библиотек)
- Содержит две части:
 - Package declaration
 - Package body

Объявление ENTITY

```
ENTITY <Имя_Entity> IS
```

Generic declarations

Port declarations

```
END <Имя_Entity>; (Версия 1076-1987)
```

```
END ENTITY <Имя_Entity>; (Версия 1076-1993)
```

- Аналогия: символ
- Generic declarations
 - Используется для параметризации модели
- Port declarations
 - Используется для описания входов и выходов модели

Entity: объявления GENERIC

```
ENTITY <имя_entity> IS
```

```
  GENERIC (
```

```
    CONSTANT tplh, tphl: time := 5ns ;
```

```
    -- слово CONSTANT можно не указывать
```

```
    tphz, tplz: TIME := 3 ns ;
```

```
    default_value : INTEGER := 1 ;
```

Синтаксис: <Модификатор>

имя_объекта:<тип>:=<начальное значение>;

Entity: объявление внешних сигналов

```
ENTITY <имя_entity> IS  
  Generic declarations
```

```
  PORT (  
    SIGNAL clk, clr: IN BIT;  
    -- слово SIGNAL можно не указывать  
    q: OUT BIT;  
  ) ;
```

```
END ENTITY <имя_entity>;
```

Architecture

- Аналогия: схема
- Описывает функциональную и временную модель
- Должна быть ассоциирована с ENTITY
- ENTITY может ассоциироваться с несколькими архитектурами
- Процессы внутри архитектуры выполняются параллельно
- Архитектурные стили
 - Поведенческое описание
 - RTL
 - Функциональное (без определения задержек)
 - Структурное (Netlist)
 - На уровне gates
 - Смешанное описание

Архитектура

ARCHITECTURE <Имя_Архитектуры> **OF** <имя_Entity> **IS**

-- Декларации

SIGNAL temp: INTEGER := 1;

CONSTANT load: **BOOLEAN** := true ;

TYPE states **IS** (S1, S2, S3, S4) ;

-- Декларации компонентов ;

-- Декларации подтипов ;

-- Декларации атрибутов ;

-- Спецификации атрибутов ;

-- Декларации подпрограмм ;

-- Спецификации подпрограмм ;

BEGIN

Определение процессов ;

Параллельные вызовы процедур ;

Параллельные присваивания сигналов ;

Инстанцирование и привязка компонентов ;

Операторы **GENERATE**

END ARCHITECTURE <имя_архитектуры> ;

Configuration

- Используется для установления связей внутри проекта
 - Связывание ENTITY и ARCHITECTURE
 - Связывание COMPONENT и ENTITY+ARCHITECTURE
- Широко используется при моделировании
 - Предоставляет гибкие возможности при сравнении альтернативных дизайнов

```
CONFIGURATION <имя_конфигурации> OF <имя_ENTITY> IS  
  FOR <имя_архитектуры>  
  END FOR ;  
END CONFIGURATION <имя_конфигурации> ;
```

Собираем все вместе

```
ENTITY cmpl_sig IS
```

```
  PORT (
```

```
    a, b, sel: IN BIT ;
```

```
    x, y, z : OUT BIT
```

```
  );
```

```
END ENTITY cmpl_sig ;
```

```
ARCHITECTURE logic OF cmpl_sig IS
```

```
BEGIN
```

```
  -- простое присваивание сигнала
```

```
  x <= ( a AND NOT sel) OR ( b AND sel) ;
```

```
  -- условное присваивание
```

```
  y <= a WHEN sel='0' ELSE b ;
```

```
  -- параметрическое присваивание
```

```
  WITH sel SELECT
```

```
    z <= a WHEN '0',
```

```
    b WHEN '1',
```

```
    '0' WHEN OTHERS ;
```

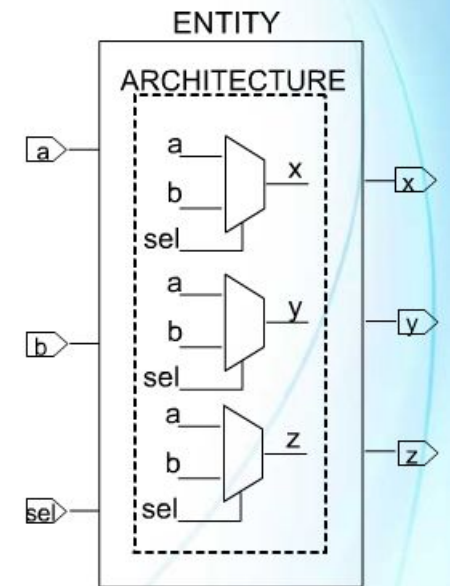
```
END ARCHITECTURE logic ;
```

```
CONFIGURATION cmpl_sig_conf OF cmpl_sig IS
```

```
  FOR logic
```

```
  END FOR ;
```

```
END CONFIGURATION cmpl_sig_conf ;
```



Packages

- Пакеты предоставляют удобную возможность для сохранения и повторного использования кода
- Пакет состоит из:
 - Декларации пакета (обязательная часть)
 - Деклараций типов
 - Деклараций подпрограмм
 - Тела пакета (может отсутствовать)
 - Определение подпрограмм
- Язык VHDL имеет два встроенных пакета:
 - Standard
 - TEXTIO

Пример использования пакета

```
LIBRARY IEEE ;  
USE IEEE.std_logic_1164.all ;
```

```
PACKAGE filt_cmp IS  
  TYPE state_type IS (idle, tap1, tap2, tap3, tap4) ;  
  FUNCTION compare (variable a,b : integer) RETURN boolean  
;
```

```
PACKAGE BODY filt_cmp IS  
  FUNCTION compare ( variable: a,b : INTEGER ) IS  
    VARIABLE temp : BOOLEAN ;  
  BEGIN  
    IF a < b THEN  
      temp := true ;  
    ELSE  
      temp := false ;  
    END IF ;  
    RETURN temp ;  
  END FUNCTION compare ;  
END PACKAGE BODY filt_cmp ;
```


Libraries

- Библиотека объединяет от одного до нескольких пакетов
- Библиотеки ресурсов
 - Стандартные пакеты
 - Пакеты IEEE
 - Пакеты производителя (Xilinx, Altera и т.п.)
 - Любые другие внешние библиотеки на которые ссылается проект
- Рабочая библиотека
 - Библиотека, в которой размещается результат компиляции текущего проекта

Использование пакетов и библиотек

- Все пакеты должны быть скомпилированы
- Неявное использование библиотек
 - Библиотеки `WORK` и `STD` не требуют специальных объявлений
- Оператор `LIBRARY`
 - Определяет имя библиотеки которую мы собираемся использовать
 - Использует символическое имя директории с файлами библиотеки
 - Определяется настройками проекта
- Оператор `USE`
 - Определяет конкретный пакет или/и объект который мы будем использовать

Стандартные библиотеки

■ Библиотека **STD**

- Содержит следующие пакеты
 - **Standard** (Типы: **Bit**, **Boolean**, **Integer**, **Real**, **Time** и функции для поддержки этих типов)
 - **Textio** (Файловые операции)
- Встроенные библиотеки

Пакет Standard

■ Тип **BIT**

- Принимает одно из двух значений '0' или '1'
SIGNAL a_tmp: **BIT** ;
- Типы данных оканчивающиеся на **_VECTOR** позволяют работать с массивами
SIGNAL temp : **BIT_VECTOR** (3 **DOWNTO** 0) ;
SIGNAL temp : **bit_vector** (0 **TO** 3) ;

■ Тип **BOOLEAN**

- (**false**, **true**)

■ Тип **INTEGER**

- Положительные и отрицательные целые числа
SIGNAL int_tmp: **INTEGER** ; -- 32-битовое целое
SIGNAL int_tmp1: **INTEGER RANGE** 0 to 255 ; -- 8 бит

Другие типы из пакеты Standard

■ Тип **NATURAL**

- Целое в диапазоне от 0 до 2^{32}

■ Тип **POSITIVE**

- Целое в диапазоне от 1 до 2^{32}

■ Тип **CHARACTER**

- ASCII символы

■ Тип **STRING**

- Массив символов

■ Тип **TIME**

- Интервал времени с единицами измерения (ps, us, ns, ms, sec, min, hr)

Стандартные библиотеки

■ Библиотека **IEEE** ;

- Содержит следующие пакеты:
 - **std_logic_1164** (тип `std_logic` и функции для работы с ним)
 - **std_logic_arith** (арифметические функции)
 - **std_logic_signed** (арифметические операции со знаком)
 - **std_logic_unsigned** (беззнаковые арифметические операции)

Типы данных из пакета `std_logic_1164`

■ Тип **STD_LOGIC**

- 9-значная логика
 - '1' – лог. 1
 - '0' – лог. 0
 - 'X' – неопределенность
 - 'Z' – выс.импеданс
 - '-' – Don't care
- Поддержка нескольких источников сигнала

- 'H' – слабая единица
- 'L' – слабый ноль
- 'W' – слабая неопределенность

■ Тип **STD_ULOGIC**

- То же что и `STD_LOGIC` но без поддержки нескольких источников

Основы языка VHDL

Моделирование цифровых систем



Основные конструкции для моделирования

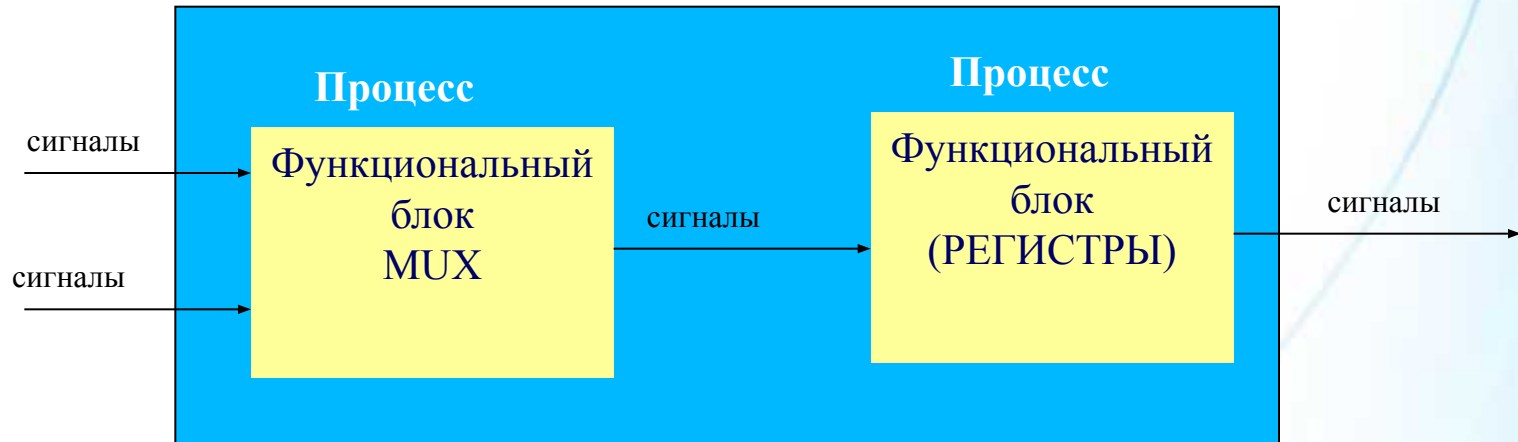
- Константы
- Сигналы
- Операторы
- Присваивание сигналов
- Процессы
- Последовательные операторы
- Переменные
- Определяемые пользователем типы

Константы

- Присваивают имя константе
- Объявление константы
 - **CONSTANT** <имя_константы> : <тип> := <значение> ;
- Константа не может изменять свое значение
- Повышает читаемость кода
- Упрощает переносимость

Сигналы

- Сигналы описывают физические соединения (проводники) между процессами (функциями)
- Сигналы могут быть объявлены в пакетах, Entity или в архитектуре

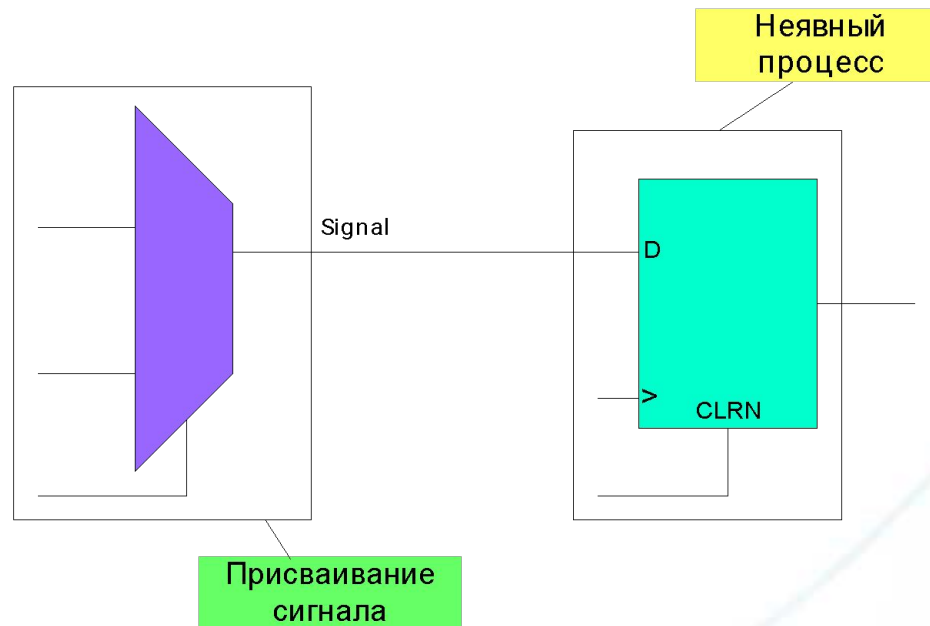


Присваивание значение сигналам

- **SIGNAL** temp: `std_logic_vector` (7 DOWNT0 0)
- Присваивание всех битов:
`temp <= "10101010" ;`
`temp <= x"AA" ;`
- Один бит:
`temp(7) <= '1' ;`
- Группа битов:
`temp(7 downto 4) <= "1010" ;`

Присваивание сигналов

- Присваивание сигналов осуществляется с помощью оператора \leftarrow
- Присваивание сигнала подразумевает создание **неявного** процесса (функции) с элементом памяти

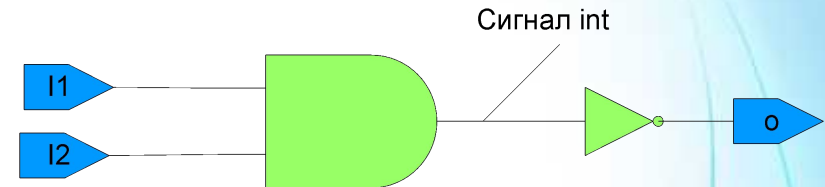


Операторы VHDL

Тип оператора	Обозначение оператора
Операторы сравнения	= /= < <= > >=
Сложение и вычитание	+ -
Мультипликативные операции	* / mod rem
Другие	** abs &

Сигналы для межсоединений

```
ENTITY simp IS
  PORT (
    i1, i2 : IN BIT ;
    o: OUT BIT
  );
END ENTITY simp ;
ARCHITECTURE logic OF simp IS
  SIGNAL int : BIT ;
BEGIN
  int <= i1 AND i2 ;
  o <= NOT int ;
END ARCHITECTURE logic ;
```



Перегрузка операторов

- VHDL определяет арифметические и логические функции только для встроенных типов данных (определенных в стандартных пакетах)
- Как использовать такие функции с другими типами данных?
 - **Перегрузка операторов** – определение арифметических и логических функций для других типов данных
- Операторы перегружаются путем определения функции с именем соответствующего оператора

Функции и пакеты перегрузки операций

- Пакеты содержащие следующие операторы могут быть найдены в библиотеке LIBRARY IEEE
 - `std_logic_arith` (арифметические функции)
 - `std_logic_signed` (знаковые арифметические функции)
 - `std_logic_unsigned` (беззнаковые арифметические функции)
 - `numeric_std` (знаковые и беззнаковые функции)
- Например пакет `std_logic_unsigned` определяет некоторые из этих функций

```
FUNCTION "+" (l: std_logic_vector; r:std_logic_vector) return std_logic_vector ;
```

```
FUNCTION "+" (l: std_logic_vector; r:INTEGER) return std_logic_vector ;
```

```
FUNCTION "+" (l: INTEGER; r:std_logic_vector) return std_logic_vector ;
```

```
FUNCTION "+" (l: std_logic_vector; r:std_logic) return std_logic_vector ;
```

```
FUNCTION "+" (l: std_logic; r:std_logic_vector) return std_logic_vector ;
```

Использование перегруженных операторов

```
Library IEEE ;  
USE IEEE.std_logic_1164.ALL ;  
USE IEEE.std_logic_unsigned.ALL ;
```

Entity overload IS

```
PORT (  
  a: IN std_logic_vector (4 DOWNTO 0) ;  
  b: IN std_logic_vector (4 DOWNTO 0) ;  
  sum: OUT std_logic_vector (4 DOWNTO 0) ;
```

Параллельное присваивание сигналов

- Используется для присваивания значений сигналу с использованием различных выражений
- Подразумевает создание неявного процесса, выполняемого параллельно с другими процессами
 - (список чувствительности такого процесса – все переменные справа от знака присваивания)
- Три типа оператора параллельного присваивания:
 - Простой оператор присваивания
 - Условный оператор присваивания
 - Case-оператор присваивания

Простой оператор присваивания

- Формат: `<имя_сигнала> <= <выражение> ;`
- Пример: `qa <= r or t ;`
`qb <= (qa and not (g xor h)) ;`

Условный оператор присваивания

- Формат:

```
<имя_сигнала> <= <выражение1> when <условие1> else  
    <выражение2> when <условие2> else  
    ...  
    <выражениеn> when <условиен> else  
    <выражениеn+1>
```

- Пример:

```
qa <= a WHEN sela = '1' ELSE  
      b WHEN selb = '1' ELSE  
      c ;
```

Неявный
процесс

Оператор присваивания WITH

- Формат:

```
WITH <выражение> SELECT  
<имя_сигнала> <= <выражение1> WHEN <условие1>,  
                <выражение2> WHEN <условие2>,  
                <выражениеn> WHEN OTHERS ;
```

- Пример:

```
WITH sel SELECT  
q <= a WHEN "00",  
   b WHEN "01",  
   c WHEN "10",  
   d WHEN OTHERS ;
```

Неявный
процесс

Задержка в операторе присваивания

- В операторе присваивания можно использовать задержку
 - Два типа задержек:
 - Инерционная задержка (по умолчанию)
 - Импульс длина которого короче указанного значения не будет передан
 - Пр. $a \leq b$ **AFTER 10 ns** ;
 - Транспортная задержка
 - любой импульс будет передан, несмотря на его длительность
 - Пр. $a \leq$ **TRANSPORT b AFTER 10 ns** ;

Явное описание процесса

- Оператор процесса выполняется до тех пор пока не встретит оператор WAIT или список чувствительности процесса.
 - Список чувствительности подобен оператору WAIT в конце процесса
 - Процесс может иметь несколько операторов WAIT
 - Процесс не может иметь одновременно и список чувствительности и оператор WAIT
- Процесс содержит последовательные операторы
- Параллельное исполнение
 - Архитектура может включать в себя несколько процессов
 - Все процессы исполняются параллельно

```
метка: PROCESS (<список_чув.>
    <объявления констант>
    <объявления типов>
    <объявления переменных>
BEGIN
    <последовательный оператор>
    ...
    <последовательный оператор>
END PROCESS ;
```

Примеры процессов

```
proc1: PROCESS (a,b)  
BEGIN
```

```
-- последовательные операторы
```

```
END PROCESS ;
```

```
proc1: PROCESS  
BEGIN
```

```
-- последовательные операторы
```

```
WAIT ON (a,b) ;
```

```
END PROCESS ;
```

Последовательные операторы

- Последовательные операторы
 - Простое присваивание сигнала
 - Оператор IF-THEN
 - Оператор CASE
 - Оператор цикла
 - Оператор WAIT

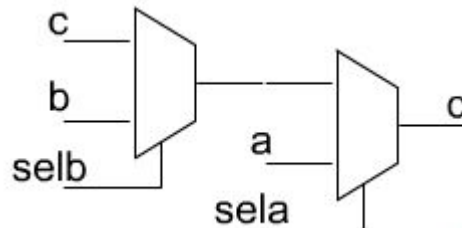
Оператор IF-THEN

■ Формат

```
IF <условие> THEN
    <последовательность
    операторов>
ELSIF <условие2> THEN
    <последовательность
    операторов>
ELSE
    <последовательность
    операторов>
END IF ;
```

■ Пример

```
PROCESS (sela, selb, a, b, c)
BEGIN
    IF sela='1' THEN
        q<=a ;
    ELSIF selb='1' THEN
        q <= b ;
    ELSE
        q <= c ;
    END IF ;
END PROCESS ;
```



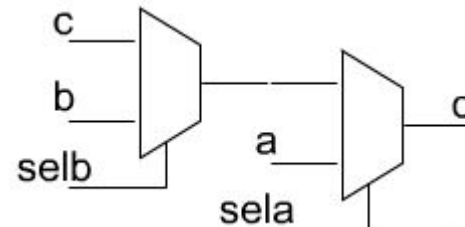
Оператор CASE

■ Формат

```
CASE <выражение> IS
  WHEN <условие1> =>
    <посл. операторы>
  WHEN <условие2> =>
    <посл. операторы>
  ...
  WHEN OTHERS => -- (опц.)
    <посл. операторы>
END CASE ;
```

■ Пример

```
PROCESS (sela, selb, a, b, c)
BEGIN
  CASE sel IS
    WHEN "00" =>
      q<=a ;
    WHEN "01" =>
      q<=b ;
    WHEN
      "10" =>
      q<=c ;
    WHEN OTHERS =>
      q<=d ;
  END CASE ;
END PROCESS ;
```



Последовательные операторы цикла

- Оператор LOOP
 - повторяется бесконечно пока не встретится оператор EXIT
- Оператор WHILE
 - Выход по условию в конце цикла
- Оператор FOR
 - Цикл на основе счетчика

[метка] LOOP

-- последовательные операторы

NEXT метка **WHEN** ... ;

EXIT метка **WHEN** ... ;

END LOOP ;

WHILE <условие> LOOP

-- последовательные операторы

END LOOP ;

FOR <идент.> IN <range> LOOP

-- послед. операторы

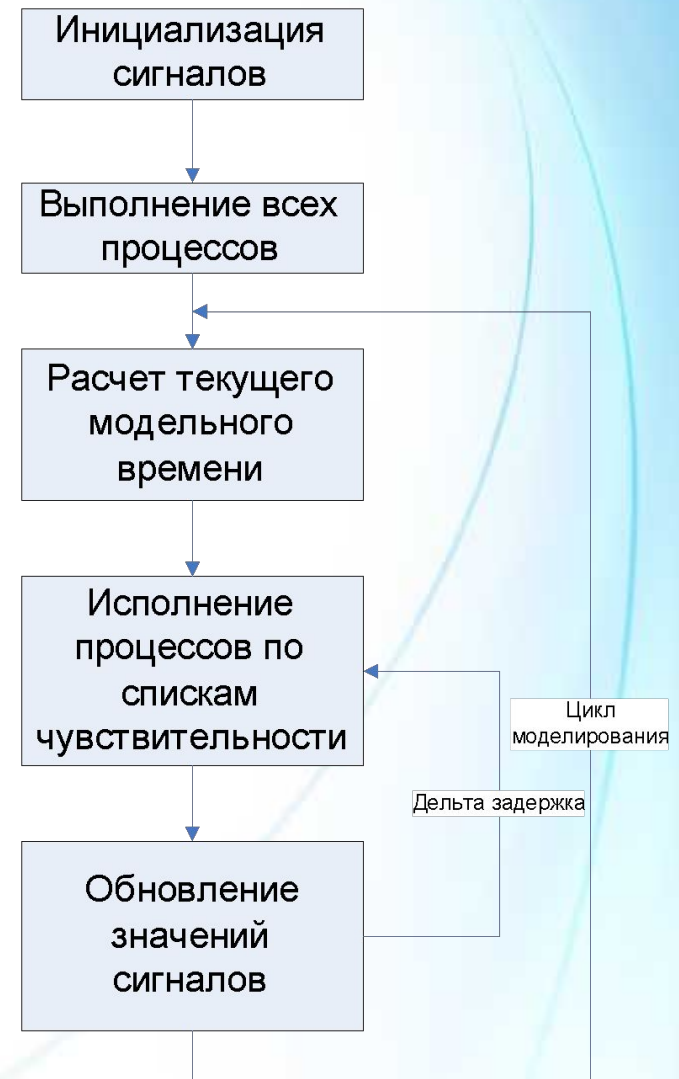
END LOOP ;

Оператор WAIT

- **WAIT ON <сигнал>**
 - Приостанавливает исполнение до события, связанного с сигналом
`WAIT ON a,b ;`
- **WAIT UNTIL <логическое_выражение>**
 - Останавливает исполнение до момента, когда выражение станет истиной
`WAIT UNTIL (int < 100) ;`
- **WAIT FOR <интервал>**
 - приостанавливает выполнение на указанный интервал
`WAIT FOR 20 ns ;`
- **Смешанный WAIT**
`WAIT UNTIL (a='1') FOR 5 us ;`

VHDL-симуляция

- Событие(Event) – любое изменение сигнала
- Цикл моделирования
 - модельное время
 - Дельта-задержка
 - Фаза выполнения процесса
 - Фаза обновления сигналов
- Цикл моделирования заканчивается когда выполнены все процессы и обновлены все сигналы



Эквивалентные функции

```
LIBRARY IEEE ;
  USE IEEE.Std_logic_1164.ALL ;
ENTITY simp IS
  PORT (
    a,b : IN std_logic ;
    y : OUT std_logic ;
  ) ;
END ENTITY simp ;
ARCHITECTURE logic OF simp IS
  SIGNAL c : std_logic ;
BEGIN
  c <= a AND b ;
  y <= c ;
END ARCHITECTURE logic ;
```

```
LIBRARY IEEE ;
  USE IEEE.Std_logic_1164.ALL ;
ENTITY simp IS
  PORT (
    a,b : IN std_logic ;
    y : OUT std_logic ;
  ) ;
END ENTITY simp ;
ARCHITECTURE logic OF simp IS
  SIGNAL c : std_logic ;
BEGIN
  Process1: PROCESS(a,b)
  BEGIN
    c <= a AND b ;
  END PROCESS process1 ;
  Process2: PROCESS(c)
  BEGIN
    y <= c ;
  END PROCESS Process2 ;
END ARCHITECTURE logic ;
```

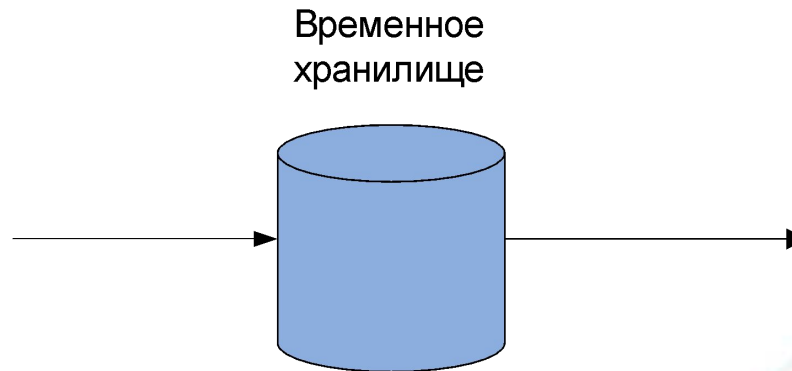
Неэквивалентные функции

```
LIBRARY IEEE ;
  USE IEEE.Std_logic_1164.ALL ;
ENTITY simp IS
  PORT (
    a,b : IN std_logic ;
    y : OUT std_logic ;
  ) ;
END ENTITY simp ;
ARCHITECTURE logic OF simp IS
  SIGNAL c : std_logic ;
BEGIN
  c <= a AND b ;
  y <= c ;
END ARCHITECTURE logic ;
```

```
LIBRARY IEEE ;
  USE IEEE.Std_logic_1164.ALL ;
ENTITY simp IS
  PORT (
    a,b : IN std_logic ;
    y : OUT std_logic ;
  ) ;
END ENTITY simp ;
ARCHITECTURE logic OF simp IS
  SIGNAL c : std_logic ;
BEGIN
  PROCESS( a, b)
    c <= a AND b ;
    y <= c ;
  END PROCESS
END ARCHITECTURE logic ;
```

Объявление переменных

- Переменные объявляются внутри процесса
- Для присваивания используется оператор **:=**
- Объявление переменных
 - VARIABLE <имя>: <Тип_данных> := <выражение> ;
 - VARIABLE temp: std_logic_vector (7 DOWNT0 0) ;
- Обновление переменной происходит немедленно
- Не вносит задержку



Присваивание значений переменным

```
VARIABLE temp : std_logic_vector ( 7 DOWNT0 0 ) ;
```

- Все биты

```
temp := "10101010" ;
```

```
temp := x"AA" ;
```

- Один бит

```
temp(7) := '1' ;
```

- Группа битов

```
temp(7 DOWNT0 4) := "1010" ;
```

- Один бит: апостроф ‘

- Несколько бит (строка): кавычки “

Эквивалентные функции

```
LIBRARY IEEE ;  
  USE IEEE.Std_logic_1164.ALL ;  
ENTITY simp IS  
  PORT (  
    a,b : IN std_logic ;  
    y : OUT std_logic ;  
  ) ;  
END ENTITY simp ;  
ARCHITECTURE logic OF simp IS  
  SIGNAL c : std_logic ;  
BEGIN  
  c <= a AND b ;  
  y <= c ;  
END ARCHITECTURE logic ;
```

```
LIBRARY IEEE ;  
  USE IEEE.Std_logic_1164.ALL ;  
ENTITY simp IS  
  PORT (  
    a,b : IN std_logic ;  
    y : OUT std_logic ;  
  ) ;  
END ENTITY simp ;  
ARCHITECTURE logic OF simp IS  
  SIGNAL c : std_logic ;  
BEGIN  
  PROCESS( a, b)  
    VARIABLE c: std_logic ;  
  BEGIN  
    c <= a AND b ;  
    y <= c ;  
  END PROCESS  
END ARCHITECTURE logic ;
```

Сигналы против переменных

	Сигналы	Переменные
Присваивание	сигнал \leftarrow выражение	переменная $:=$ выражение
Назначение	Соединение элементов дизайна	Локальное хранение данных
Область видимости	Глобальная (обмен между процессами)	Локальная (только внутри процесса)
Поведение	Значение обновляется в конце дельта-цикла	Обновление происходит незамедлительно

Определяемые пользователем типы

- Массивы Arrays
- Перечислимые типы данных

Массив (Array)

- Создает двумерный тип данных
 - Созданный тип необходимо использовать при объявлении констант, сигналов или переменных такого типа
- Используется для резервирования памяти и размещения тестовых векторов или данных
- Объявление типа памяти

```
TYPE <имя_типа> IS ARRAY (<диапазон_целых>) OF  
    <тип_данных_элемента> ;
```

Пример использования массива

```
ARCHITECTURE logic OF my_memory IS
```

```
    TYPE mem IS ARRAY (0 to 63) OF std_logic_vector (7 DOWNT0 0) ;  
    -- создается новый тип данных «массив» с именем mem который  
    -- использует 64 адресных позиции по 8 бит каждая  
  
    SIGNAL mem_64x8_a, mem_64x8_b : mem ;  
    -- создается 2 64x8бит массива
```

```
BEGIN
```

```
    mem_64x8_a(12) <= x"A4" ;  
    mem_64_8_b(50) <= "11110000" ;
```

```
END ARCHITECTURE logic ;
```

Enumerated Data Type

- Позволяет перечислить все значения определяемого типа данных
 - Используется при определении констант, сигналов или переменных этого типа
- Используется для
 - Повышения читаемости кода
 - При описании конечных автоматов
- Объявление перечислимого типа данных

TYPE <имя_типа> **IS** (перечисление значений через запятую) ;

```
TYPE enum IS (idle, fill, heat_w, wash, drain) ;  
SIGNAL dshwshr_st : enum ;  
  
...  
drain_led <= '1' WHEN dshwsher_st = drain ELSE '0' ;
```



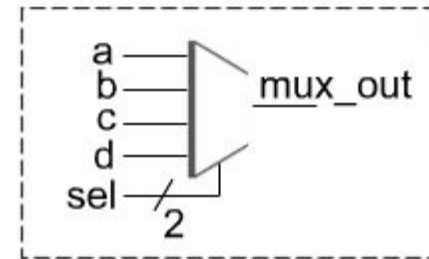
Основы языка VHDL

Синтез цифровых систем

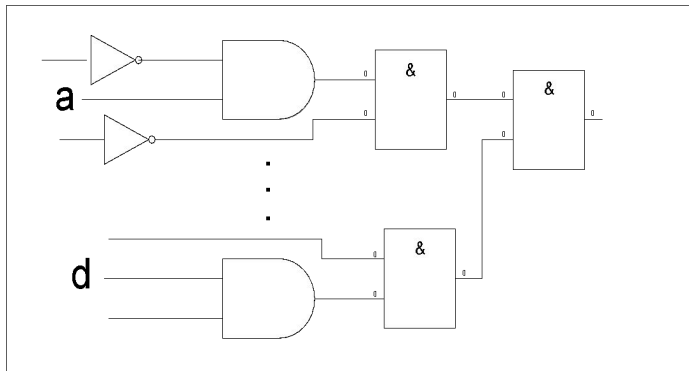


RTL синтез

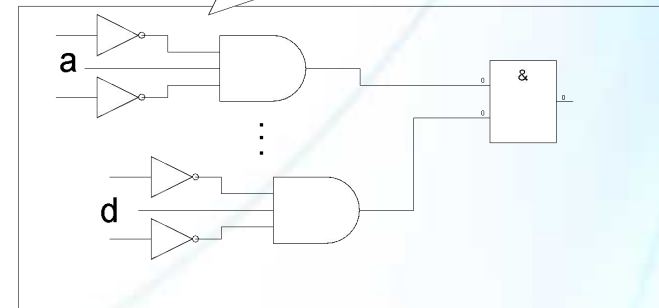
```
Process( a,b,c,d,sel )  
begin  
  case (sel) is  
    when "00" => mux_out <= a;  
    when "01" => mux_out <= b;  
    when "10" => mux_out <= c;  
    when "11" => mux_out <= d;  
  end case ;  
end process;
```



Трансляция

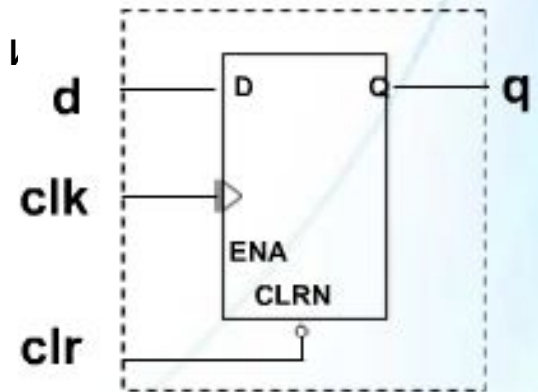
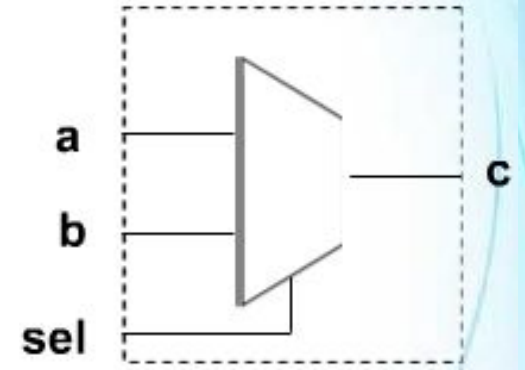


Оптимизация



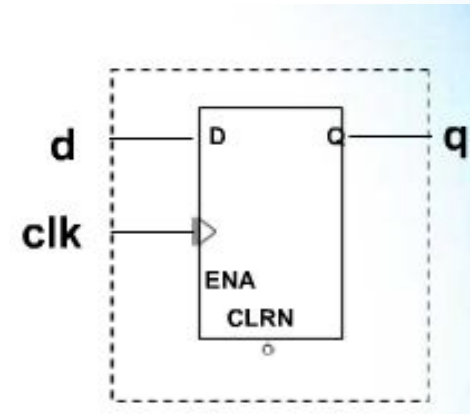
Два типа синтеза процесса

- Комбинаторный процесс
 - Список чувствительности включает все входы логических элементов
- Пример
 - **PROCESS(a,b,sel)**
- Последовательный процесс
 - Чувствителен только к сигналам тактирования и к управляющим сигналам
- Пример
 - **PROCESS(clr, clk)**



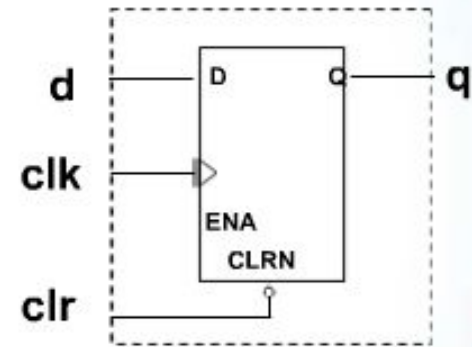
D-триггер на основе функции rising_edge

```
LIBRARY IEEE ;  
  USE IEEE.Std_logic_1164.ALL ;  
ENTITY dff_b IS  
  PORT (  
    clk,d : IN std_logic ;  
    q : OUT std_logic  
  ) ;  
END ENTITY dff_b ;  
ARCHITECTURE rtl OF dff_b IS  
  SIGNAL c : std_logic ;  
BEGIN  
  PROCESS(clk)  
  BEGIN  
    IF rising_edge(clk) THEN  
      q <= d ;  
    END IF ;  
  END PROCESS ;  
END ARCHITECTURE logic ;
```



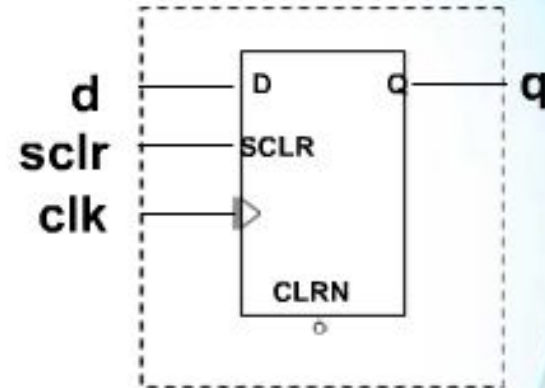
D-триггер с асинхронным сбросом

```
LIBRARY IEEE ;
  USE IEEE.Std_logic_1164.ALL ;
ENTITY dff_aclr IS
  PORT (
    d,clk,clr : IN std_logic ;
    q : OUT std_logic
  ) ;
END ENTITY dff_aclr ;
ARCHITECTURE rtl OF dff_aclr IS
  BEGIN
  PROCESS(clk)
  BEGIN
    IF clr='0' THEN
      q <= '0' ;
    ELSIF rising_edge(clk) THEN
      q <= d ;
    END IF ;
  END PROCESS ;
END ARCHITECTURE logic ;
```



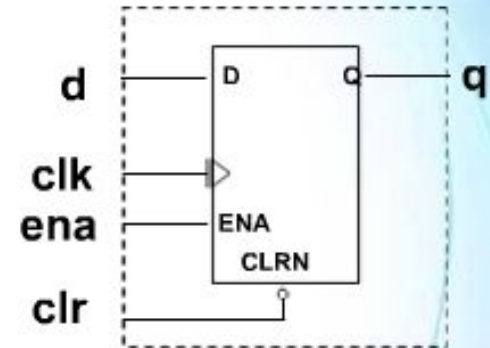
D-триггер с синхронным сбросом

```
LIBRARY IEEE ;  
  USE IEEE.Std_logic_1164.ALL ;  
ENTITY dff_aclr IS  
  PORT (  
    d,clk,clr : IN std_logic ;  
    q : OUT std_logic  
  ) ;  
END ENTITY dff_aclr ;  
ARCHITECTURE rtl OF dff_aclr IS  
  BEGIN  
    PROCESS(clk)  
    BEGIN  
      IF rising_edge(clk) THEN  
        IF clr='0' THEN  
          q <= '0' ;  
        ELSE  
          q <= d ;  
        END IF ; END IF ;  
    END PROCESS ;  
  END ARCHITECTURE logic ;
```



D-триггер с асинхронным сбросом и управлением синхронизацией

```
LIBRARY IEEE ;
  USE IEEE.Std_logic_1164.ALL ;
ENTITY dff_aclr_ena IS
  PORT (
    d,clk,clr, ena : IN std_logic ;
    q : OUT std_logic
  ) ;
END ENTITY dff_aclr_ena ;
ARCHITECTURE rtl OF dff_aclr_ena IS
  BEGIN
  PROCESS(clk,clr)
  BEGIN
    IF clr='0' THEN
      q <= '0' ;
    ELSIF rising_edge(clk) THEN
      IF ena='1' THEN
        q <= d ;
      END IF ;
    END IF ;
  END PROCESS ;
END ARCHITECTURE rtl ;
```



Синтез регистра

- Присваивание сигнала внутри оператора IF-THEN с условием проверки сигнала тактирования приводит к синтезу регистра

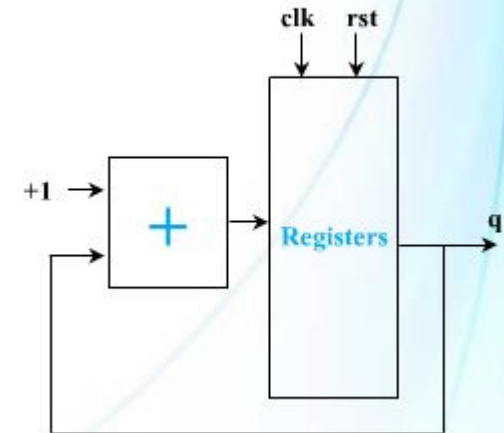
```
PROCESS (clk)
BEGIN
  IF rising_edge( clk ) THEN
    q <= d ;
  END IF ;
END PROCESS ;
```

Синтез счетчика

```
LIBRARY IEEE ;
  USE IEEE.Std_logic_1164.ALL ;
  USE IEEE.Std_logic_unsigned.ALL ;

ENTITY counter IS
  PORT (
    clk, rst : IN std_logic ;
    q : OUT std_logic_vector (15 DOWNTO 0)
  ) ;
END ENTITY counter ;

ARCHITECTURE logic OF counter IS
  SIGNAL tmp_q : std_logic_vector (15 DOWNTO 0) ;
BEGIN
  PROCESS(clk, rst)
  BEGIN
    IF rst='0' THEN
      tmp_q <= (OTHERS => '0') ;
    ELSIF rising_edge(clk) THEN
      tmp_q <= tmp_q + 1 ;
    END IF ;
  END PROCESS ;
  q <= tmp_q ;
END ARCHITECTURE ;
```



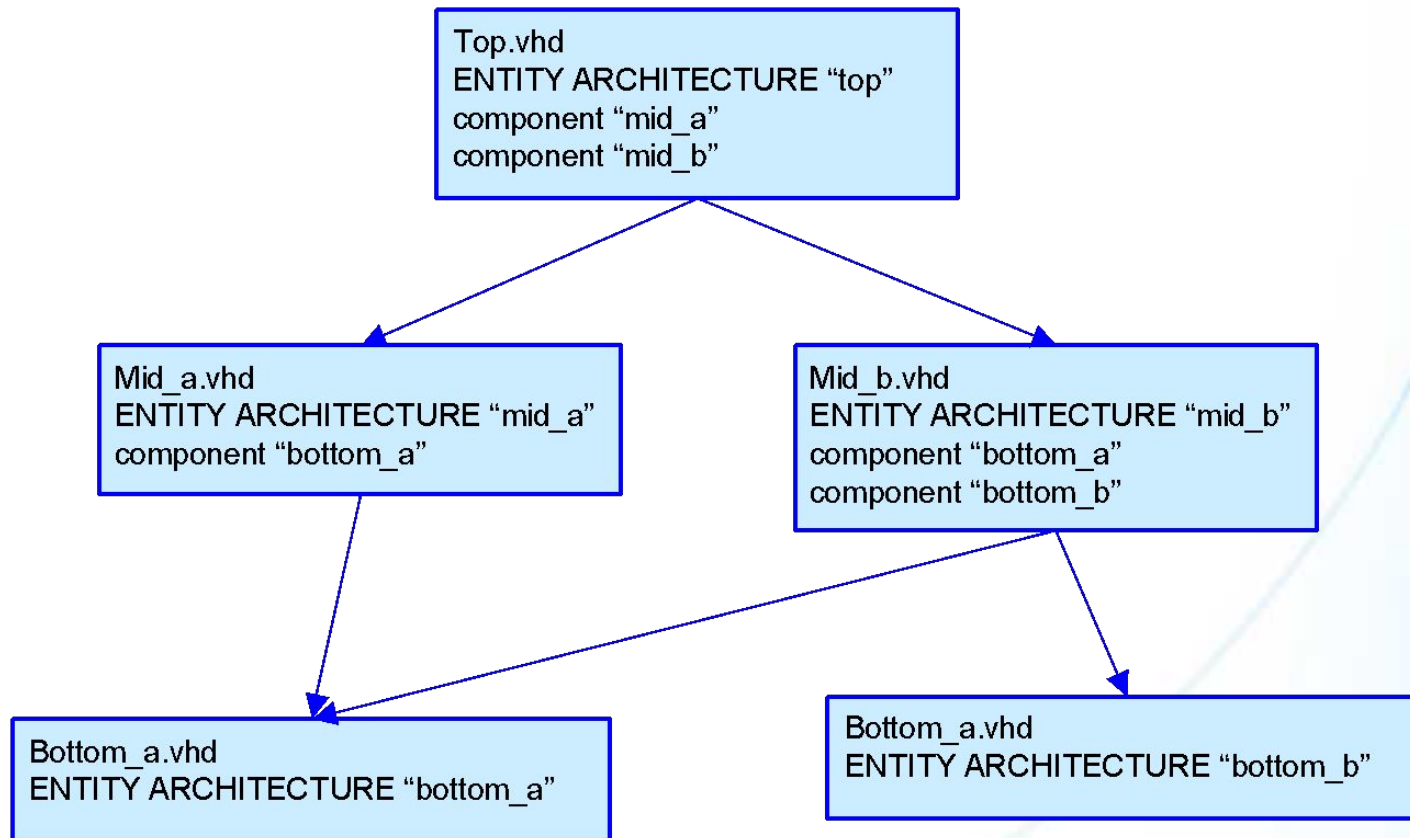
Основы языка VHDL

Структурное описание



Иерархический дизайн – дизайн в нескольких файлах

- Иерархический дизайн использует объявление (Declaration) и размещение (Instantiation) компонентов



Объявление и размещение компонентов

- Объявление компонента используется для указания типов данных и портов другого ENTITY

```
COMPONENT <имя_подключаемого_entity>  
  PORT (  
    <имя_порта> : <тип_порта> <тип_данных> ;  
  
    <имя_порта> : <тип_порта> <тип_данных>  
  ) ;  
END COMPONENT ;
```

- Размещение компонента – параллельный оператор, используемый для вставки и привязки компонента в текущую архитектуру

```
<имя_размещения> : <имя_подключаемого_entity>  
  PORT MAP (<имя_порта_подключаемого_entity> => <сигнал>  
  
            ...  
            <имя_порта_подключаемого_entity> => <сигнал>  
  ) ;
```

Объявление и размещение компонентов

```
LIBRARY IEEE ;  
  USE IEEE.Std_logic_1164.ALL ;  
  ENTITY tollv IS  
    PORT (  
      tclk, tcross, tnickel, tdime, tquarter : IN std_logic ;  
      tgreen, tred : OUT Std_logic  
    ) ;  
  END ENTITY tollv ;
```

```
ARCHITECTURE tollv_arch OF tollv IS
```

```
  COMPONENT tollc  
    PORT (  
      clk, cross, nickel, dime, quarter : IN std_logic ;  
      green, red : OUT std_logic  
    ) ;
```

```
END COMPONENT ;
```

```
BEGIN
```

```
U1: tollc PORT MAP (clk=>tclk, cross=>tcross, nickel=>tnickel, dime=>tdime,  
  quarter => tquarter, green => tgreen, red => red )
```

```
END ARCHITECTURE tollv_arch ;
```


Основы языка VHDL

*Дополнительные операторы языка
VHDL*



Оператор GENERATE

```
<метка_группы>: FOR <индекс> IN <диапазон> GENERATE  
  <метка_компонента>: <имя_компонента>  
    [ PORT MAP ]  
  END GENERATE ;
```

```
COMPONENT register_4 IS  
  PORT (  
    data_in: IN std_logic_vector (3 DOWNT0 0) ;  
    data_out : OUT std_logic_vector (3 DOWNT0 0) ;  
    clk: IN std_logic  
  ) ;  
END COMPONENT ;
```

BEGIN

```
registers: for i from 0 to 3 generate  
  Reg: register_4  
  port map (  
    data_in=>data_16_in((i+1)*4-1 downto i*4),  
    data_out=> data_16_out ((i+1)*4-1 downto i*4),  
    clk=>clk16 ) ;  
  END GENERATE Registers ;
```

```
END ARCHITECTURE Registers_Arch ;
```

Типы данных



ALTERA®

Спасибо за внимание

