# Логичко пројектовање

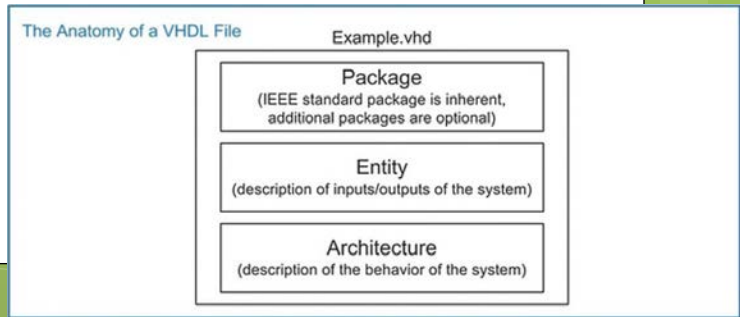
Предавање 1/12

**Основе VHDL језика (I део)**

**Опис комбинационих мрежа у VHDL- у**

# VHDL конструкције (1/2)

- VHDL датотека
  - VHDL дизајн описује један систем у једној датотеци. Датотека има суфикс *.vhd.
  - Унутар датотеке су два дела која описују систем: ентитет и архитектура. Ентитет описује интерфејс према систему (тј. улазе и излазе) док архитектура описује понашање.
  - Функционалност VHDL-а (нпр. оператори, врсте сигнала, функције) су дефинисане у пакету.
  - Пакети су груписани у оквиру библиотека.
  - На следећој слици је приказан графички приказ VHDL датотеке.
- IEEE пакет
  - IEEE дефинише основни скуп функционалности за VHDL стандард у пакету. Овај пакет се налази у библиотеци која се зове IEEE.
  - Библиотека и укључивање пакета се наводи на почетку VHDL датотеке пре ентитета и архитектура.
  - Додатне функционалности се могу додати у VHDL укључивањем других пакета, али су сви пакети засновани на основној функционалности дефинисаној у стандардном пакету.
  - Није потребно експлицитно навести да дизајн користи стандардни пакет IEEE јер је својствен коришћењу VHDL-а.
  - Све функције описане у овом поглављу су за IEEE стандардни пакет.



# VHDL конструкције (2/2)

- VHDL основна правила
  - VHDL не разликује мала и велика слова.
  - Такође, свака VHDL додела, дефиниција или декларација се завршава тачком и зарезом (;).
  - Пребацивање додела, дефиниција или декларација у нову линију је дозвољено.
  - Празне линије се могу користити да VHDL буде читљивији.
  - Коментари у VHDL-у су означени са две цртице (тј. --) на почетку реда и коментар се наставите до краја реда.
  - Сва кориснички дефинисана имена у VHDL-у морају да почињу алфабетским словом, а не бројем.
  - Кориснички дефинисаним именима није дозвољено да буду иста као било која VHDL кључна реч.
- Интерне ознаке
  - Овај предмет садржи многе дефиниције синтаксе у ВХДЛ-у. Следеће ознаке ће се користити у целом градиву

| | |
|---|---|
| **bold** | = VHDL keyword, use as is |
| *italics* | = User-defined name |
| <> | = A required characteristic such as a data type and input/output |

-

# Типови података (1/5)

- У VHDL-у, сваком сигналу, константи, променљивој и функцији може бити додељен тип података.
- IEEE стандардни пакет пружа разне унапред дефинисане типове података.
- Неки типови података се могу синтетизовати, док су други само за моделирање апстрактних понашања.
- Следе најчешће коришћени типови података у VHDL-у
- Набројиви типови података

| Type | Values that the type can take on |
|---|---|
| **bit** | {0, 1} |
| **boolean** | {false, true} |
| **character** | {“any of the 256 ASCII characters defined in ISO 8859-1”} |

- Тип података са опсегом вредности

| Type | Values that the type can take on |
|---|---|
| **integer** | Whole numbers between −2,147,483,648 to +2,147,483,647 |
| **real** | Fractional numbers between $-1.7e^{38}$ to $+1.7e^{38}$ |

# Типови података (2/5)

- Физички тип - онај који садржи и вредност и јединицу мере.

| Type | Values that the type can take on |
|---|---|
| **time** | Whole numbers between −2,147,483,648 to +2,147,483,647 |

| | | |
|---|---|---|
| (unit relationships) | **fs** | (femtosecond, $10^{-15}$), base unit |
| | **ps** = 1000 fs | (picosecond, $10^{-12}$) |
| | **ns** = 1000 ps | (nanosecond, $10^{-9}$) |
| | **µs** = 1000 ns | (microsecond, $10^{-6}$) |
| | **ms** = 1000 µs | (millisecond, $10^{-3}$) |
| | **s** = 1000 ms | (second) |
| | **min** = 60 s | (minute) |
| | **h** = 60 min | (hour) |

# Типови података (3/5)

- Векторски тип - онај који се састоји од линеарног низа скаларних типова.
- Величина векторског типа је дефинисана укључивањем максималног индекса, кључне речи “downto” и минималнног индекса.
- На пример, ако је сигнал BUS_A типа бит_bit_vector(7 downto 0), креира се вектор од 8 скаларних вредности где је свака типа бит. Крајњи леви скалар би имао индекс 7, а крајњи десни скалар би имао индекс 0. Сваки од појединачних скалара унутар вектора се може приступити навођењем индекса броја у загради. На пример BUS_A(0) би приступио скалару на крајњој десној позицији. Индекси не морају увек да имају минималну вредност од 0.

| Type | Construction |
|---|---|
| **bit_vector** | A linear array of type bit |
| **string** | A linear array of type character |

- Корисничко дефинисани набројиви тип података - онај у коме је наведено име типа од стране корисника поред свих могућих вредности које тип може да узме.

```
type name is (value1, value2, …);
```

Example:

```
type traffic_light is (red, yellow, green);
```

# Типови података (4/5)

- Низ - садржи више елемената истог типа.
  - Елементи унутар низа могу бити скаларни или вектори.
  - Да би се користио низ, мора се декларисати нови тип који дефинише конфигурацију низа.
  - Сигнали могу бити декларисани да буду овог типа.
  - Опсег низа мора бити дефинисан у декларацији низа. Опсег је специфициран целим бројевима (мин. и макс.) и било који употребом кључних речи “downto” или “to”.

```
type name is array (<range>) of <element_type>;
```

Example:

```
type block_8x16 is array (0 to 7) bit_vector(15
downto 0);
signal my_array : block_8x16;
```

# Типови података (5/5)

- Подтип - ограничена верзија или подскуп другог типа.
  - Подтипови су кориснички дефинисани типови података, иако је неколико често коришћених подтипова унапред дефинисано у стандардни пакет.
  - Најчешће коришћених подтипова (NATURAL и POSTIVE) који су дефинисани у стандардни пакет: садржи више елемената истог типа.

```
subtype name is <type> range <min> to <max>;
```

Example:

```
subtype NATURAL is integer range 0 to 255;
subtype POSTIVE is integer range 1 to 256;
```

-

# Библиотеке и пакети

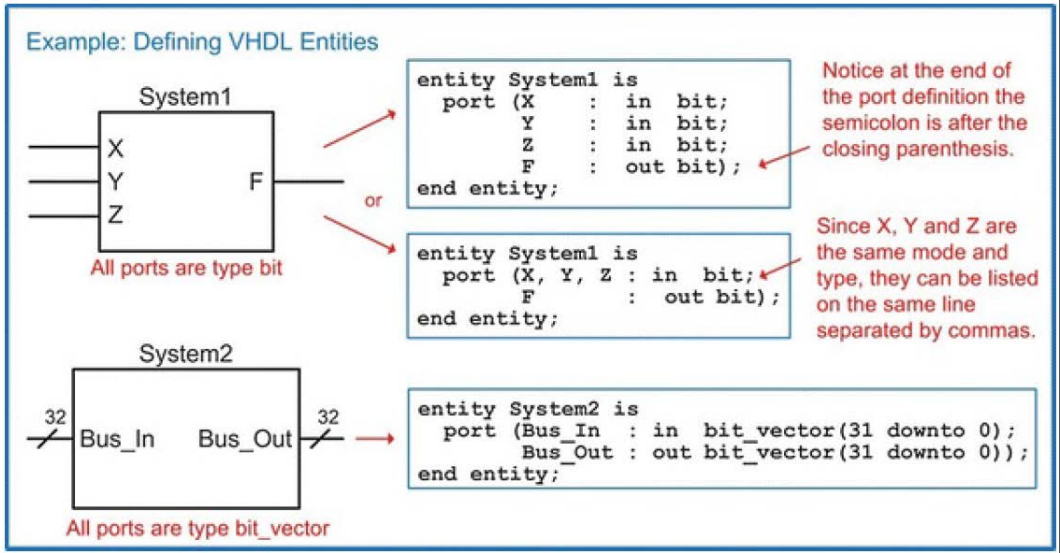
- Као што је раније поменуто, IEEE стандардни пакет се подразумева када се користи VHDL, међутим можемо га користити као пример како укључити пакете у VHDL.
- Кључна реч “library” се користи да означи да ће пакети бити додати у VHDL дизајн из наведене библиотеке.
- Иза овога следи кључна реч “use” и назив пакета из библиотеке.
- Синтакса назива пакета има три поља одвојена тачкама. Прво поље је назив библиотеке. Друго поље је назив пакета. Треће поље је специфична функционалностпакет која треба да буде укључена.
- Ако треба користити све функционалности пакета, онда иде кључна реч “all” која се користи у трећем пољу.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_textio.all;
```

# Ентитет

- Ентитет у VHDL-у описују улазе и излазе система (портови).
- Сваки порт мора имати своје име, режим и тип. Име је дефинисано од стране корисника.
- Режим описује правац преноса података кроз порт и може да узме вредности “in”, “out”, “inout” и “buffer”. Тип је један од горе описаних правних типова података.
- Имена портова са истим режимом и типом се може навести у истом реду одвојеном зарезима.

```
entity entity_name is
    port (port_name : <mode> <type>;
port_name : <mode> <type>);
end entity;
```



# Архитектура (1/4)

- Архитектура у VHDL-у описује понашање система. Постоје бројне технике за описивање понашања у VHDL-у које обухватају више нивоа апстракције.
- Архитектура је место где је већина дизајнерских радова спроведено.

```
architecture architecture_name of <entity associated
with> is

-- user-defined enumerated type
declarations    (optional)
-- signal declarations                    (optional)
-- constant declarations
(optional)
-- component declarations
(optional)

begin

-- behavioral description of the system goes here

end architecture;
```

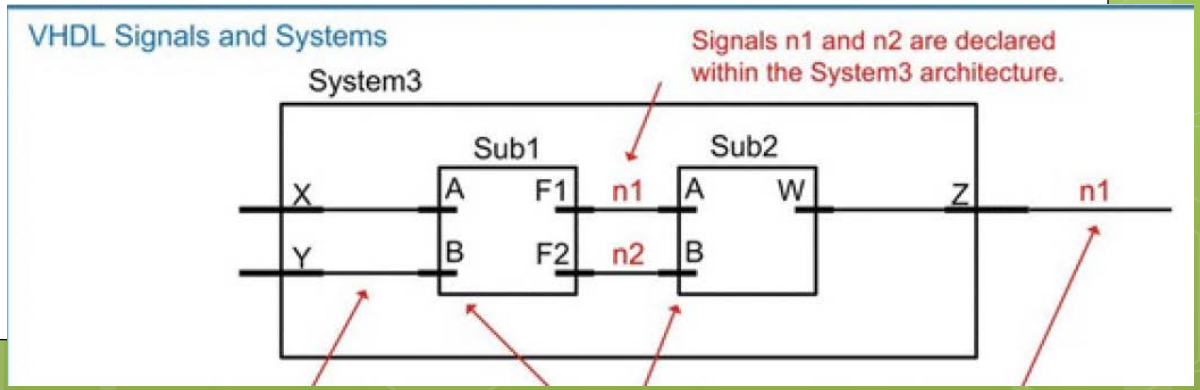
# Архитектура (2/4)

- Декларација сигнала
  - Сигнал који се користи за интерне везе унутар система је декларисан у архитектури.
  - Сваки сигнал мора бити декларисан са типом. Сигнал може бити самокористи се за прављење веза сличних типова. Сигнал је декларисан са кључном речи “signal” праћен кориснички дефинисаним именом, двотачком и типом.
  - Сигнали сличног типа могу се декларисати у истом реду одвојеном зарезом. Сви типови података се могу користити за сигнале. Сигнали представљају жице унутар система, тако да немају правац ни режим.
  - Сигнали не могу имати иста имена као портови у систему.

```
signal name : <type>;
```

Example:

```
signal node1 : bit;
signal a1, b1 : integer;
signal Bus3 : bit_vector (15 downto 0);
signal C_int : integer range 0 to 255;
```



# Архитектура (3/4)

- Декларација константи
  - Константа је корисна за представљање количине која ће се користити више путау архитектури.
  - Име константе се може користити у целој архитектури.

```
constant constant_name : <type> := <value>;
```

Example:

```
constant BUS_WIDTH : integer := 32;
```

Example:

```
signal BUS_A : bit_vector (BUS_WIDTH-1 downto 0);
```

# Архитектура (4/4)

- Декларација компоненти
  - Компонента је термин који се користи за VHDL подсистем који је инстанциран унутар система вишег нивоа.
  - Ако ће се компонента користити у систему, она мора бити декларисан у архитектури пре наредбе “begin”.
  - Дефиниције порта компоненте морају одговарати дефиницијама порта ентитета подсистема. Као такве, оне се обично копирају директно из описа VHDL ентитета система нижег нивоа.
  - Једном декларисана, компонента може бити инстанцирана након наредбе “begin” у архитектури онолико пута колико је потребно.

```
component component_name
    port (port_name : <mode> <type>;
port_name : <mode> <type>);
end component;
```

# Моделовање конкурентне функционалности у VHDL-у

- Важно је запамтити да је VHDL језик описа хардвера, а не програмски језик.
- У програмском језику, линије кода се извршавају узастопно како се појављују у изворној датотеци.
- У VHDL-у линије кода представљају понашање стварног хардвера. Као резултат, сви додељени сигнали се подразумевано извршавају истовремено осим ако није другачије назначено.
- Све операције у VHDL-у морају бити сличних типова и резултат мора бити истог типа као и улази операција.

# VHDL оператори (1/3)

- Оператор доделе
  - VHDL користи “<=“ за све доделе сигнала и “:=“ за све промењиве и иницијализације.
  - Ови оператори доделе раде на свим типовима података. Додела има смер са десна на лево.

```
   F1 <= A;      -- F1 and A must be the same size and
type
   F2 <= '0';       -- F2 is type bit in this example
   F3 <= "0000";      -- F3 is type bit_vector(3 downto
0) in this example
   F4 <= "hello";     -- F4 is type string in this
example
   F5 <= 3.14;      -- F5 is type real in this example
   F6 <= x"1A";      -- F6 is type bit_vector(7 downto
0), x"1A" is in HEX
```

# VHDL оператори (2/3)

- Логички оператори
  - Ови оператори раде на типовима “bit”, “bit_vector” и “Boolean”.
  - За операције типа “bit_vector”, улазни вектори морају бити исте величине и одвијају се у “bit-wise” режиму. На пример, ако две 8-битне магистрале које се зову BusA и BusB имају операцијау AND, AND операција се на пример изводи посебно на BusA(0) и BusB(0).
  - Редослед приоритета у VHDL-у је другачији од оног у Буловој алгебри. Оператор NOT има већи приоритет од свих осталих оператора. Све остало логичкi оператори имају исти приоритет.

| Operator | Operation |
|---|---|
| **not** | Logical negation |
| **and** | Logical AND |
| **nand** | Logical NAND |
| **or** | Logical OR |
| **nor** | Logical NOR |
| **xor** | Logical exclusive-OR |
| **xnor** | Logical exclusive-NOR |

Example:

```
F1 <= not A;
F2 <= B and C;
```

Example:

```
   F3 <= not D nand E;          -- D will be
complemented first, the result
                                                --
will then be NAND'd with E, then the
                          -- result will be assigned to
F3
   F4 <= not (F or G);          -- the parentheses take
precedence so
                          -- F will be OR'd with G
first, then                                     --
complemented, and then assigned to F4.

   F5 <= H nor I nor J;     -- logic operations can
have any number of
                           -- inputs.

   F6 <= K xor L xnor M;      -- XOR and XNOR have the
same priority so with
                           -- no parentheses given,
the logic operations
                           -- will take place on the
signals from
                           -- left to right. K will
be XOR'd with L first,
                           -- then the result will be
XNOR'd with M.
```

# VHDL оператори (3/3)

- Нумерички оператори

| | |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| **mod** | Modulus |
| **rem** | Remainder |
| **abs** | Absolute value |
| ** | Exponential |

- Релациони оператори

| Operator | Returns true if the comparison is: |
|---|---|
| = | Equal |
| /= | Not equal |
| < | Less than |
| <= | Less than or equal |
| > | Greater than |
| >= | Greater than or equal |

Шифт оператори

| Operator | Operation |
|---|---|
| **sll** | Shift left logical |
| **srl** | Shift right logical |
| **sla** | Shift left arithmetic |
| **sra** | Shift right arithmetic |
| **rol** | Rotate left |
| **ror** | Rotate right |

```
   A <= B srl 3;              -- A is assigned the
result of a logical shift                        -- right
3 times on B.
```

Оператор конкатенације

Example:

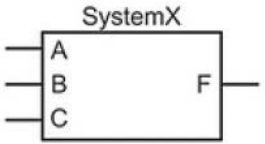
```
  Bus1 <= "11" & "00";     -- Bus1 must be 4-bits and
will be assigned
             -- the value "1100"

  Bus2 <= BusA & BusB;     -- If BusA and BusB are 4-
bits, then Bus2
                                 -- must be 8-bits.

  Bus3 <= '0' & BusA;      -- This attaches a leading
'0' to BusA. Bus3
                                 -- must be 5-bits
```

# Конкурентна додела сигнала са логичким операторима

Implement the following truth table using concurrent signal assignments with logical operators.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

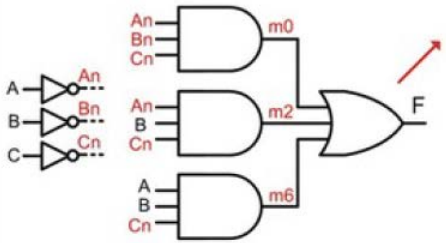
First, let's design the entity. Let's call the entity *SystemX*. The entity will have three inputs (A, B, C) and one output (F). We'll use the type bit for all inputs/outputs so that this will synthesize directly into real circuitry.




```
entity SystemX is
   port (A, B, C  :  in  bit;
         F        :  out bit);
end entity;
```

Now we design the architecture. We can create a canonical sum of products logic expression for this truth table using minterms.

$$F = \sum_{A,B,C}(0,2,6) = A'\cdot B'\cdot C' + A'\cdot B\cdot C' + A\cdot B\cdot C'$$




```
architecture SystemX_arch of SystemX is

  signal  An, Bn, Cn : bit;
  signal  m0, m2, m6 : bit;

begin

  An <= not A;              -- NOT's
  Bn <= not B;
  Cn <= not C;

  m0 <= An and Bn and Cn; -- AND's
  m2 <= An and B  and Cn;
  m6 <= A  and B  and Cn;

  F  <= m0 or m2 or m6;    -- OR

end architecture;
```

# Условна додела сигнала

Example: Modeling Logic using Conditional Signal Assignments

Implement the following truth table using a conditional signal assignment.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

```
entity SystemX is
   port (A, B, C  :  in  bit;
         F        :  out bit);
end entity;
```

We can implement the entire truth table in its current form using a conditional signal assignment. While this is a verbose approach, it is sometimes more readable.

```
architecture SystemX_arch of SystemX is

begin

  F <= '1' when (A='0' and B='0' and C='0') else
       '0' when (A='0' and B='0' and C='1') else
       '1' when (A='0' and B='1' and C='0') else
       '0' when (A='0' and B='1' and C='1') else
       '0' when (A='1' and B='0' and C='0') else
       '0' when (A='1' and B='0' and C='1') else
       '1' when (A='1' and B='1' and C='0') else
       '0' when (A='1' and B='1' and C='1');

end architecture;
```

```
architecture SystemX_arch of SystemX is

begin

  F <= '1' when (A='0' and B='0' and C='0') else
       '1' when (A='0' and B='1' and C='0') else
       '1' when (A='1' and B='1' and C='0') else
       '0';

end architecture;
```

# Селекциона додела сигнала

Example: Modeling Logic using Selected Signal Assignments

Implement the following truth table using a selected signal assignment.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

We can implement the entire truth table in its current form using a selected signal assignment. Since we are basing our output values on three separate scalar inputs, we need to concatenate them into a vector so that the new vector name can be used as the input in the selected signal assignment. We'll first declare a new signal called "ABC" of type bit_vector(2 downto 0). After the begin

```
entity SystemX is
   port (A, B, C  :  in  bit;
         F        :  out bit);
end entity;
```

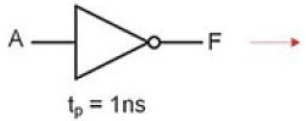
```
architecture SystemX_arch of SystemX is

  signal ABC : bit_vector(2 downto 0);

begin

  ABC <= A & B & C;

  with (ABC) select
     F <= '1' when "000",
          '0' when "001",
          '1' when "010",
          '0' when "011",
          '0' when "100",
          '0' when "101",
          '1' when "110",
          '0' when "111";

end architecture;
```

```
architecture SystemX_arch of SystemX is

  signal ABC : bit_vector(2 downto 0);

begin

  ABC <= A & B & C;

  with (ABC) select
     F <= '1' when "000",
          '1' when "010",
          '1' when "110",
          '0' when others;

end architecture;
```

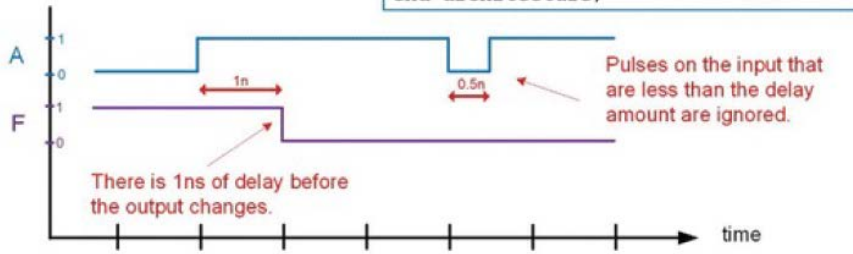
```
architecture SystemX_arch of SystemX is

  signal ABC : bit_vector(2 downto 0);

begin

  ABC <= A & B & C;

  with (ABC) select
     F <= '1' when "000"|"010"|"110",
          '0' when others;

end architecture;
```

# Одложена додела сигнала

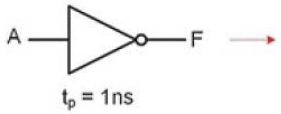
Example: Modeling Logic using Delayed Signal Assignments (Inertial Delay Model)



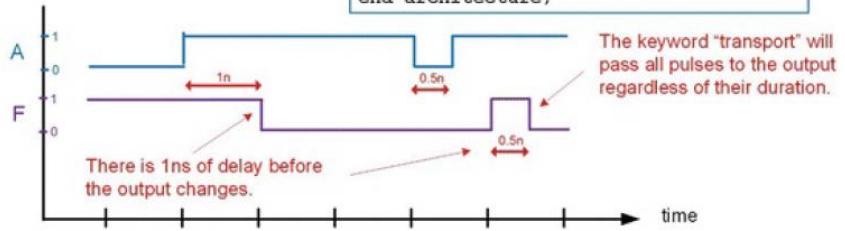

```
entity INV1 is
   port (A : in  bit;
         F : out bit);
end entity;

architecture INV1_arch of INV1 is

begin

  F <= not A after 1ns;

end architecture;
```




Example: Modeling Logic using Delayed Signal Assignments (Transport Delay Model)




```
entity INV2 is
   port (A : in  bit;
         F : out bit);
end entity;

architecture INV2_arch of INV2 is

begin

  F <= transport not A after 1ns;

end architecture;
```

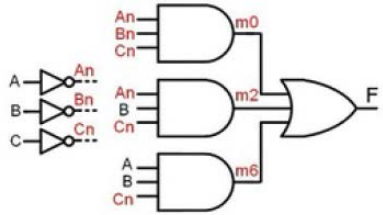


# Структурни дизајн коришћењем компоненти

Example: Modeling Logic using Structural VHDL (Explicit Port Mapping)

Implement the following truth table with structural VHDL using lower level sub-systems for the basic gates. We will assume that VHDL designs have been completed for the inverter, AND gate, and OR gate. The entities for these designs are provided.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |




The basic gate designs can be declared as components in our system and then instantiated in order to describe the sum of products logic diagram above.

```
entity INV1 is
 port (A : in  bit;
       F : out bit);
end entity;
```

```
entity AND3 is
 port (A,B,C : in  bit;
       F     : out bit);
end entity;
```

```
entity OR3 is
 port (A,B,C : in  bit;
       F     : out bit);
end entity;
```

```
entity SystemX is
   port (A, B, C  :  in  bit;
         F        :  out bit);
end entity;

architecture SystemX_arch of SystemX is

  signal  An, Bn, Cn : bit;   -- declare signals
  signal  m0, m2, m6 : bit;

  component INV1              -- declare INV1
     port (A : in  bit;
           F : out bit);
  end component;

  component AND3              -- declare AND3
     port (A,B,C : in  bit;
           F     : out bit);
  end component;

  component OR3
     port (A,B,C : in  bit;   -- declare OR3
           F     : out bit);
  end component;

begin

  U1 : INV1 port map (A=>A, F=>An);
  U2 : INV1 port map (A=>B, F=>Bn);
  U3 : INV1 port map (A=>C, F=>Cn);

  U4 : AND3 port map (A=>An, B=>Bn, C=>Cn, F=>m0);
  U5 : AND3 port map (A=>An, B=>B,  C=>Cn, F=>m2);
  U6 : AND3 port map (A=>A,  B=>B,  C=>Cn, F=>m6);

  U7 : OR3  port map (A=>m0, B=>m2, C=>m6, F=>F);

end architecture;
```

The entity is named SystemX.

Internal signals are needed to connect the sub-systems.

The three lower level sub-systems are declared as components in SystemX.

The components are instantiated and connected using explicit port mapping in order to describe the behavior of the logic diagram.

NOT's

AND's

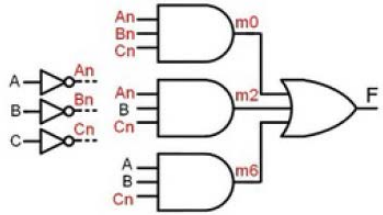
OR

# Позиционо мапирање портова

Example: Modeling Logic using Structural VHDL (Explicit Port Mapping)

Implement the following truth table with structural VHDL using lower level sub-systems for the basic gates. We will assume that VHDL designs have been completed for the inverter, AND gate, and OR gate. The entities for these designs are provided.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |




```
entity INV1 is
 port (A : in  bit;
       F : out bit);
end entity;
```

```
entity AND3 is
 port (A,B,C : in  bit;
       F     : out bit);
end entity;
```

```
entity OR3 is
 port (A,B,C : in  bit;
       F     : out bit);
end entity;
```

The basic gate designs can be declared as components in our system and then instantiated in order to describe the sum of products logic diagram above.

Example: Modeling Logic using Structural VHDL (Positional Port Mapping)

In positional port mapping the port names are not listed in the component instantiation. Instead, the signals are simply listed in the same order as the ports were defined. The signal listed first will be connected to the port defined first. The signal listed second will be connected to the port defined second, etc.

Explicit Port Mapping

```
begin

  U1 : INV1 port map (A=>A, F=>An);
  U2 : INV1 port map (A=>B, F=>Bn);
  U3 : INV1 port map (A=>C, F=>Cn);

  U4 : AND3 port map (A=>An, B=>Bn, C=>Cn, F=>m0);
  U5 : AND3 port map (A=>An, B=>B,  C=>Cn, F=>m2);
  U6 : AND3 port map (A=>A,  B=>B,  C=>Cn, F=>m6);

  U7 : OR3  port map (A=>m0, B=>m2, C=>m6, F=>F);
```

Positional Port Mapping of Same System

```
begin

  U1 : INV1 port map (A, An);
  U2 : INV1 port map (B, Bn);
  U3 : INV1 port map (C, Cn);

  U4 : AND3 port map (An, Bn, Cn, m0);
  U5 : AND3 port map (An, B,  Cn, m2);
  U6 : AND3 port map (A,  B,  Cn, m6);

  U7 : OR3  port map (m0, m2, m6, F);
```