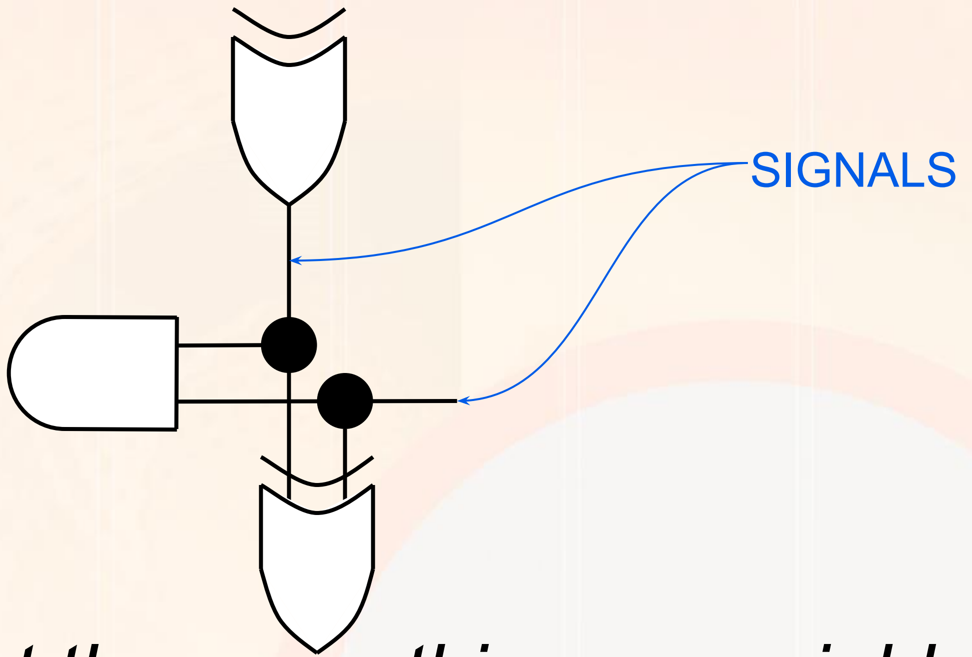
Digital System Design Lab.

Lecture 2

Salam Al-Khammasi
Dept. of ECE
Faculty of Engineering, University of Kufa
E-Mail: **salam.alshemmari@uokufa.edu.iq**

Signals

- *Signals are the basic elements of VHDL – they define the structure and behaviour of the wires connecting the circuit components, and consequently the behaviour of the components themselves.*



- *They are <u>not</u> the same thing as variables in a high level programming language!*

Signals

```
signal P,G,Cprop: STD_LOGIC;
```

- *Signals are declared using the* **SIGNAL** *keyword. Legal names (and this applies to all names in VHDL) must start with a letter (A-Z, a-z) and can contain only alphanumeric characters (A-Z, a-z, 0-9) and underscores (_).*
- *All signals have a type, which defines the* ***range and kind of values*** *they can represent. Signal types are specified at declaration (VHDL is a strongly typed language).*
- *Conversions between different types must be performed explicitly.*

Signals – single wires

- *For synthesizable VHDL (and for "simple" designs) <u>single-bit</u> signals are defined using IEEE standard 1164 types* **STD_LOGIC**:

```
type STD_LOGIC is:
     '0' - Forcing Zero      'W' - Weak Unknown
     '1' - Forcing One       'L' - Weak Zero
     'U' - Unitialized       'H' - Weak One
     'X' - Forcing Unknown     '-' - Don't Care
     'Z' - High Impedance
```

- *Only '0', '1', and (rarely) 'Z' can be directly used in synthesis. All other values are for <u>simulation and debugging</u> only.*

Signals – multiple wires (busses)

- *For synthesizable VHDL (and for "simple" designs) multi-bit signals should be defined using one of 3 possible IEEE standard types:*
  - **STD_LOGIC_VECTOR**
  - **UNSIGNED**
  - **SIGNED**
- *All these types are formally defined as 1-dimensional arrays of* **STD_LOGIC** *(i.e. any bit within the arrays can take any of the values of the* **STD_LOGIC** *type).*
- *Their **size** must be defined explicitly and, obviously, **cannot be variable**:*

```
signal A : std_logic_vector(7 downto 0);
signal B : signed(15 downto 0);
signal C : unsigned(constant downto 0);
```

Signals – multiple wires (busses)

- *Why the complexity?*
- *Consider arithmetic and comparison operations on binary code:*

```
    if (1100 > 0011) then …

  0111 + 1010 -- overflow?
```

- **`STD_LOGIC_VECTOR`** *carries no numerical connotation.* **`UNSIGNED`** *and* **`SIGNED`** *carry <u>additional information</u> that is used by the <u>synthesis tools</u>.*
- *All three types <u>represent the same hardware</u> and are often used together in a single design. Conversion between these types is very simple (more in lecture 5).*

Structure of a VHDL file

LIBRARY DECLARATIONS	`LIBRARY ieee;`
	`USE ieee.std_logic_1164.all;`

ENTITY	`ENTITY entity_name IS`
	`    PORT (……);`
	`END entity_name;`

ARCHITECTURE	`ARCHITECTURE arch_name OF entity_name IS`
	`    [declarations]`
	`BEGIN`
	`    [code]`
	`END arch_name;`

Note: comments in VHDL start with `--` ; any text after this until the end of the line is ignored in synthesis

Libraries

- *Standard Library (`STD`) - <u>Implicit</u>*
  - *Library that contains the standard VHDL functions, types and components*
- *Working Library (`work`) - <u>Implicit</u>*
  - *Library into which the unit is being compiled*
- *Resource Libraries*
  - *IEEE developed libraries*
  - *ALTERA and Xilinx (FPGA) component libraries*
  - *Personal / company-specific libraries*
  - *Etc…*
- *Libraries are structured in one or more packages*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

Required to use types *signed* and *unsigned* (i.e. for all arithmetic/ comparison operations on busses)

Libraries in VHDL

```
LIBRARY <name>;
USE <name>.<package_name>.all;
```

Structure of a VHDL file

LIBRARY DECLARATIONS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

ENTITY

```
ENTITY entity_name IS
    PORT (……);
END entity_name;
```

ARCHITECTURE

```
ARCHITECTURE arch_name OF entity_name IS
    [declarations]
BEGIN
    [code]
END arch_name;
```

Entities

- *An entity represents the interface of a circuit*

```
entity full_adder is
   port (A,B,Cin: in std_logic;
              S, Cout: out std_logic);
end full_adder;
```

- *It can be considered the main VHDL building block: equivalent to a symbol in a schematic design, it describes the interface to a hierarchical block* ***without defining behaviour***

- *It is essentially a list that specifies all input and output pins (ports) of the circuit – note that* ***output ports cannot be read inside the architecture****!*

Entities

- Syntax (note the punctuation):
  - *Keyword* **entity**, *followed by the design name, then* **is**
  - *Keyword* **port** *keyword, followed by a list of I/O signals, where the mode can be* **in** *or* **out** *(or* **buffer** *or* **inout***) and the type must be standard or defined in a library*
  - *Keyword* **end**, *followed by the design name*

```
entity entity_name is
   port( port_name(s): signal_mode signal_type;
      port_name(s): signal_mode signal_type;
      […]
      );
end entity_name;
```

```
entity full_adder is
   port (  A,B: in std_logic;
      Cin: in std_logic;
           S, Cout: out std_logic);
end full_adder;
```

Ports of the same mode and of the same type can be on a single line (but don't have to be)

Structure of a VHDL file

LIBRARY DECLARATIONS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

ENTITY

```
ENTITY entity_name IS
    PORT (……);
END entity_name;
```

ARCHITECTURE

```
ARCHITECTURE arch_name OF entity_name IS
    [declarations]
BEGIN
    [code]
END arch_name;
```

Architecture

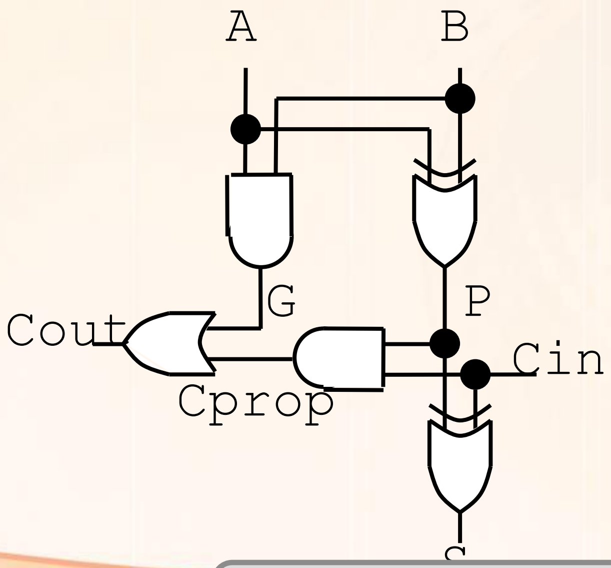
- *The architecture is the description of the circuit's functionality (behaviour).*
- *The syntax is :*

```
architecture arch_name of entity_name is
  [declarations]
begin
  [code]
end arch_name;
```

- *Where the architecture name can be anything (there can be more than one architecture for an entity… more on this later) and the entity name must be the same as in the entity declaration.*

Architecture

- *Internal signals are declared before* **begin**.
- *These are wires that are used inside the design but do not enter or leave it – all internal signals (any wire you use inside the circuit) must be explicitly declared.*
- *Of course, there might not be any, so this part is optional…*



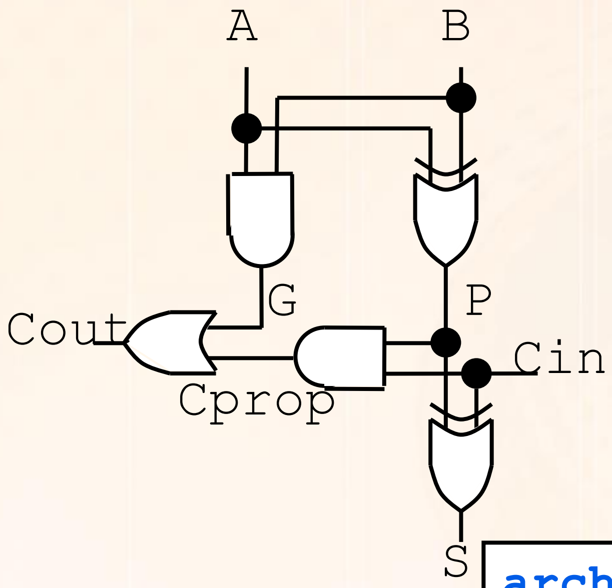
Signals of the same type can be on a single line (but don't have to be)

```
architecture arch of full_adder is
  signal P,G: STD_LOGIC;
  signal Cprop: STD_LOGIC := '0';
begin
  [behavioural or structural description]
end arch;
```

IMPORTANT: Signals can be initialized **BUT SHOULD NOT BE!**

Architecture

- *After* **begin**, *the functionality of the circuit is described, using functions, components, and operators defined in the libraries or by the designer*
- *Note that, unlike schematics, HDLs allow several different implementation styles (more on this later)*

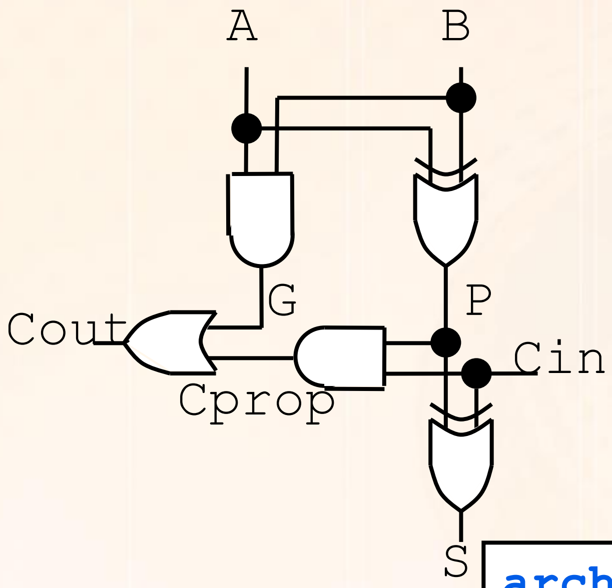


```
architecture arch of full_adder is
signal P,G,Cprop: STD_LOGIC;
begin
    G <= A and B;
    P <= A xor B;
    Cprop <= P and Cin;
    Cout <= Cprop or G;
    S <= P xor Cin;
end arch;
```

```
architecture arch of full_adder is
begin
    Cout <= ((A xor B) and Cin) or (A and B);
    S <= (A xor B) xor Cin;
end arch;
```

Architecture

- *After* **`begin`***, the functionality of the circuit is described, using functions, components, and operators defined in the libraries or by the designer*
- *Note that, unlike schematics, HDLs allow several different implementation styles (more on this later)*



```
architecture arch of full_adder is
signal P,G,Cprop: STD_LOGIC;
begin
    S <= P xor Cin;
    G <= A and B;
    P <= A xor B;
    Cprop <= P and Cin;
    Cout <= Cprop or G;
    end arch;
```

```
architecture arch of full_adder is
begin
    Cout <= ((A xor B) and Cin) or (A and B);
    S <= (A xor B) xor Cin;
end arch;
```

Architecture

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    Concurrent signal assignment
    Concurrent procedural calls
    Process statements
    Component instantiation statements
    Generate statements
END architecture_name;
```

Concurrent signal assignment

There are 3 types of assignments

1. Simple signal assignment

```
x <= a NAND b;
```

2. Conditional signal assignment

```
output <= input WHEN (ena = ‘0’)ELSE
    (OTHERS => ‘Z’);
```

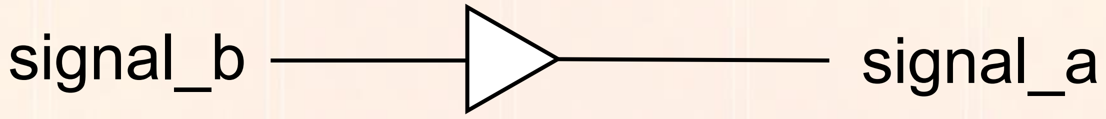
3. Selected signal assignment

```
WITH s SELECT
   f <= d0 WHEN '0',
       d1 WHEN OTHERS;
```

Assignments

- ***Assignment*:**

```
signal_a <= signal_b;
```



- *The current value of* `signal_b` *is assigned to* `signal_a`

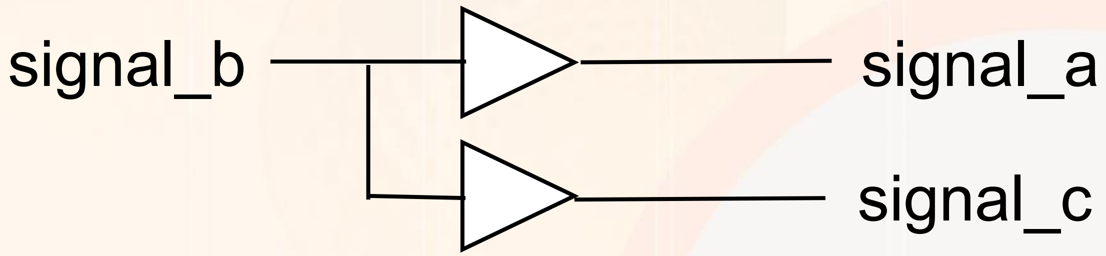
- *VHDL assignments are* ***NOT*** *the same as variable assignments in a programming language. They create a physical connection between two lines (signals) in a circuit.*

Assignments and concurrency

- *Unless the assignment is within a process (lecture 4), think of this connection simply as a wire (or buffer).*
- *There is no notion of sequence: signal assignments are concurrent statements (outside a process) and the order in which they are written is irrelevant!*

```
signal_c <= signal_b;
signal_a <= signal_b;
```

```
signal_a <= signal_b;
signal_c <= signal_b;
```

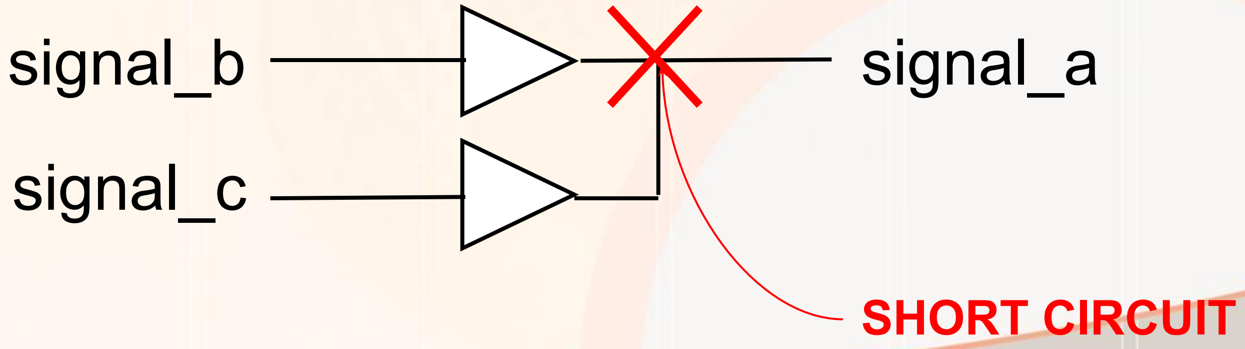


- *When the circuit is simulated, a statement is executed whenever a signal on the right hand side changes: i.e., an event on one signal leads to an event on another*

Assignments and concurrency

- *Unless the assignment is within a process (lecture 4), think of this connection simply as a wire (or buffer).*
- *But this also means that* ***two or more assignments to the same signal create a short circuit!***

```
signal_a <= signal_b;
signal_a <= signal_c;
```



Single wires

- ***Single line:*** **`STD_LOGIC`**

  Example:
  ```
  a, b : STD_LOGIC;
      a <= '0';
      a <= b;
  ```
- *Single explicit values are always between single apostrophes (`'0'`, `'1'`)*
- *Note that while STD_LOGIC can potentially have many values (`'U'`, `'W'`, `'X'`, etc.), for synthesis only `'0'`, `'1'`, and `'Z'` make any sense!*
- *Therefore, while is*
  ```
  a <= 'U';
  ```
  is a perfectly valid VHDL statement, it is NOT synthesizable.

Vectors (busses)

- ***Bus:*** `STD_LOGIC_VECTOR(n downto m)`
- *A major advantage of VHDL over schematics is that vectors (busses) are assigned bit-wise* ***almost exactly like single wires*** *- the only exception being the use of* `""` *instead of* `''`.

In theory, this could be `(0 to 11)`, but MSB first is much simpler for binary numbers.

Example: `c,d : STD_LOGIC_VECTOR(11 downto 0)`

```
c <= "000000000000"; -- must be correct length
c(11 downto 6) <= "000000";
c <= d;   -- must be same length
a(12 downto 7) <= d(5 downto 0);
```

- *The size of the bus on the left of an assignment must always be the same as the size of the bus on the right*
- *The same syntax applies to* `SIGNED` *and* `UNSIGNED`.

Example - Shifter

- *Combinatorial shift/rotation requires no logic gates*

```
entity Shift is
 port (A : in STD_LOGIC_VECTOR(7 downto 0);
    SL, SR, RL, RR : out STD_LOGIC_VECTOR(7 downto 0));
end Shift;

architecture arch of Shift is
begin
    SL(7 downto 1) <= A(6 downto 0);
    SL(0) <= '0';
    SR(6 downto 0) <= A(7 downto 1);
    SR(7) <= '0';
    RL(7 downto 1) <= A(6 downto 0);
    RL(0) <= A(7);
    RR(6 downto 0) <= A(7 downto 1);
    RR(7) <= A(0);
end arch;
```

Notation for single bits within a bus

Combinational Logic Operators

- ***Standard logic operators*** *include* **AND**, **OR**, **NAND**, **NOR**, **XOR** *and* **XNOR**. *All have the same precedence and* ***execute from left to right****. The exception is* **NOT**, *which has a* ***higher precedence****, and therefore executes before other operators in an expression*
- *Multiple operators can be applied within a single assignment*

```
architecture arch of full_adder is
begin
   Cout <= ((A xor B) and Cin) or (A and B);
   S <= (A xor B) xor Cin;
end arch;
```

- *Logical operations can be applied to arrays of the same type and length, in which case they operate bitwise*

```
architecture arch of bitwise_nand is
signal A,B,C : std_logic_vector(7 downto 0)
begin
   C <= A nand B;
end arch;
```

Concurrent signal assignment

There are 3 types of assignments

1. Simple signal assignment

```
x <= a NAND b;
```

2. Conditional signal assignment

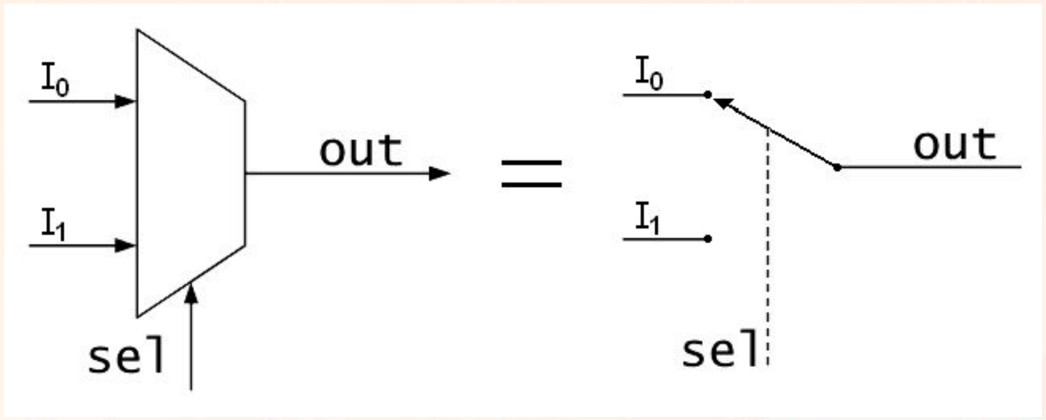
```
output <= input WHEN (ena = ‘0’)ELSE
    (OTHERS => ‘Z’);
```

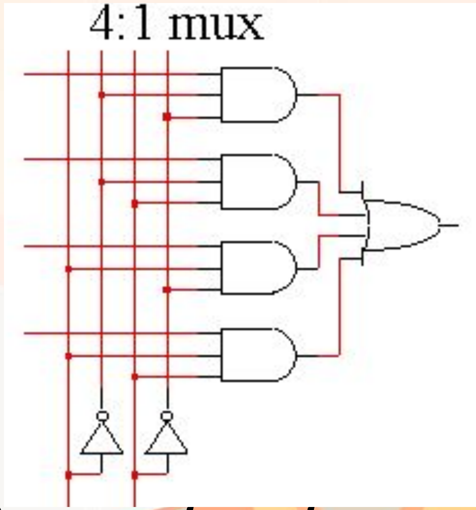
3. Selected signal assignment

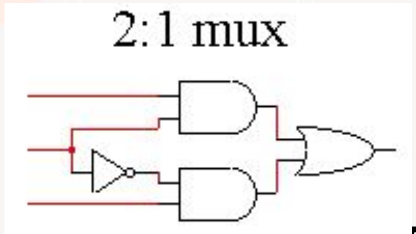
```
WITH s SELECT
   f <= d0 WHEN '0',
       d1 WHEN OTHERS;
```

Conditional statements

- *Every combinational function can be described in terms of logic gates.*
- *However, this is not always the most "natural" way to describe a function.*
- *Example: multiplexer – a vital component in logic circuit design*







- *Multiplexers <u>can</u> be described as AND/OR constructs (indeed, they <u>will</u> be implemented as such), but it is neither efficient nor clear*

Conditional statements

- *Conditional assignments are higher level constructs that give another method of description*
- *Conditional assignments have two syntactical forms:*
  - **WHEN / ELSE**

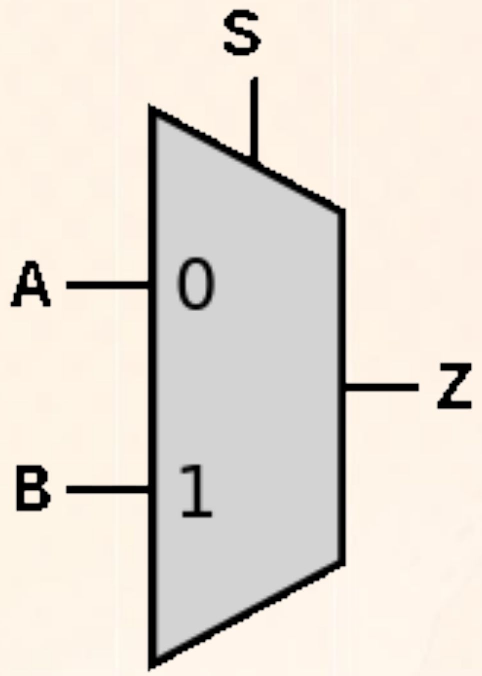
```
sig_a <=  sig_b_or_value when condition1 else
    sig_c_or_value when condition2 else
    […]
    sig_n_or_value;
```

  - **WITH / SELECT / WHEN**

```
with sig_s select
    sig_a <=  sig_b_or_value when sig_s_value1,
        sig_c_or_value when sig_s_value2,
        […]
        sig_n_or_value when others;
```

Note “catch-all” clause at end. It is necessary but does not need to be synthesizable.

Example – multiplexer 1



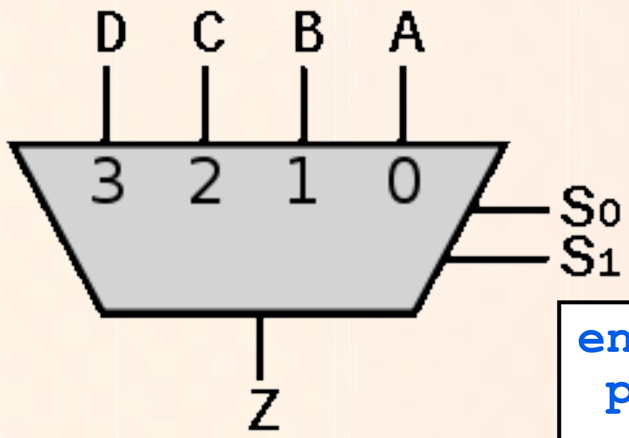
```
entity Mux2_1 is
 port (A,B,S : in STD_LOGIC;
       Z : out STD_LOGIC);
end Mux2_1;

architecture arch of Mux2_1 is
begin
    Z <=   A when S='0' else
        B when S='1' else
        'U';
end arch;
```

Note "=" : this is not an assignment

Default case (unknown inputs) Is it needed? Why? Why can it be 'U' (not synthesizable)?

- *Conditional assignment are concurrent ... if and only if one of the signals on the right changes (S, A, B)*
- *All possible values must be covered (in general, it is best to leave a "catch-all" default case)*

Example – multiplexer 2

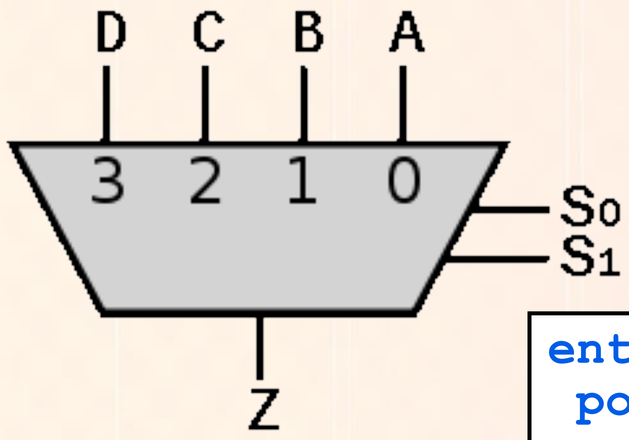


```
entity Mux4_1 is
 port (A,B,C,D : in STD_LOGIC;
    Z : out STD_LOGIC;
       S : in STD_LOGIC_VECTOR(1 downto 0));
end Mux4_1;

architecture arch of Mux4_1 is
begin
    Z <=   A when S="00" else
       B when S="01" else
       C when S="10" else
       D when S="11" else
       'U';
end arch;
```

- *Note that conditions can be arbitrarily complex and include functions*

```
O <=   A when (S(0)='0' and S(1)= '0') else [...]
```

Example – multiplexer 3



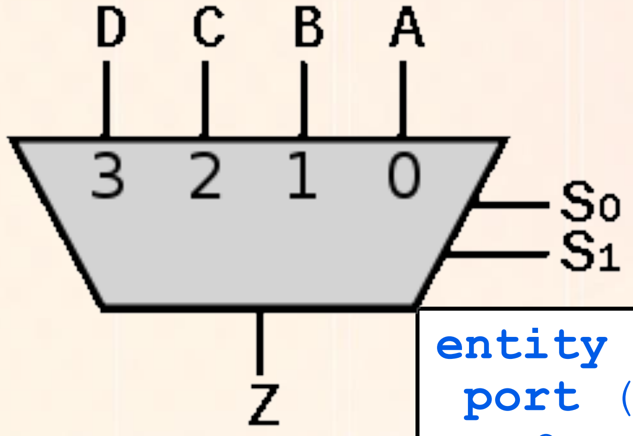
```
entity Mux4_1 is
 port (A,B,C,D : in STD_LOGIC;
    O : out STD_LOGIC;
       SEL : in STD_LOGIC_VECTOR(1 downto 0));
end Mux4_1;

architecture arch of Mux
begin
    with SEL select
       O <=   A when "00",
       B when "01",
       C when "10",
       D when "11",
       'U' when others;
end arch;
```

Note syntax of default case

- *Again, this is a* ***concurrent statement****. Will be executed if and only if one of the signals on the right changes (S, A, B, C, D)*

Example – multiplexer 4

- *Either syntax can be used, with no changes, to create multiplexers for busses, instead of single lines (obviously, the output bus has to be of the same size as the input busses).*



```
entity Mux4_1 is
 port (A,B,C,D : in STD_LOGIC_VECTOR(7 downto 0);
    O : out STD_LOGIC_VECTOR(7 downto 0);
       SEL : in STD_LOGIC_VECTOR(1 downto 0));
end Mux4_1;

architecture arch of Mux4_1 is
begin
    with SEL select
       O <=   A when "00",
       B when "01",
       C when "10",
       D when "11",
       (others => 'U') when others;
end arch;
```

More on this syntax in lecture 7

Conclusions

- *These basic ideas should be enough to get you started in VHDL & do the Lab1 – Part3, which is mostly meant to <u>familiarize you with the tools</u> (even if you probably know them already)*
- *From next week, we will start re-visiting each component of this lecture and go in more depth on each topic*