



## L3: Combinational Logic in VHDL

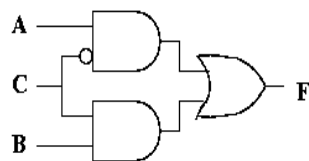


## Hazards – copied from I2



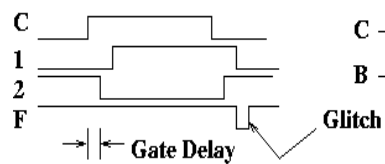
Static Hazards: Consider this function:

$$F = A \cdot \bar{C} + B \cdot C$$

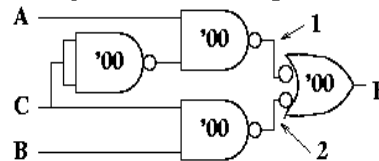


		AB			
		00	01	11	10
C	0	0	0	1	1
	1	0	1	1	0

A = B = 1



Implemented with MSI gates:

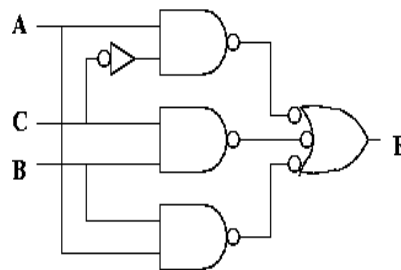




## Fixing Hazards copied from I2



The glitch is the result of timing differences in parallel data paths. It is associated with the function jumping between groupings or product terms on the K-map. To fix it, cover it up with another grouping or product term!



	AB			
	00	01	11	10
C				
0	0	0	1	1
1	0	1	1	0

$$F = A * \bar{C} + B * C + A * B$$



## VHDL Designs



- VHDL Design Descriptions consist of two parts.
  - The ENTITY describes the periphery of the design, i.e., the SIGNAL I/O.
  - The ARCHITECTURE body describes the content.
  - Both ENTITY and ARCHITECTURE have names (identifiers).
    - ENTITY names must be unique within a design.
    - ARCHITECTURES provide content for the ENTITY.
    - ARCHITECTURE names need not be unique.
      - They are used to delineate the ARCHITECTURE declaration.
      - They likely provide you (or the reader) with information.
  - ENTITIES always use a special signal, a PORT.
    - PORTS have MODEs and TYPEs. MODEs are:
      - IN
      - OUT
      - INOUT – A tri-state signal.
      - BUFFER – An output which is used internally and also has a limited fan-out.
  - VHDL is MoStLy case indePENDent. But it is best to think it is.



## Example Entity



-- A double hyphen means the rest of the line is a comment.

-- This comment is before the library and use clauses,  
-- which is needed for STD\_LOGIC.

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.ALL;
```

-- here is the entity

```
ENTITY black_box IS
```

```
PORT (clk, rst : in STD_LOGIC;
```

```
      d : in STD_LOGIC_VECTOR(7 downto 0);
```

```
      q : out STD_LOGIC_VECTOR(7 downto 0);
```

```
      co : out STD_LOGIC);
```

```
end black_box;
```

### black\_box



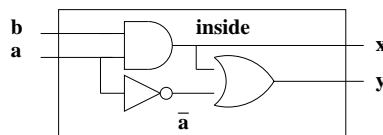
## Not Needing Mode BUFFER



### ■ We will not use mode BUFFER.

- This makes it easier to use components as VHDL is very strongly typed and one has to declare signal types that are the same as those used by the component. This eliminates confusion between modes OUT and BUFFER.
- When we want to use an output internally we will declare a signal in the ARCHITECTURE to use internally and assign the output to this internal signal.

```
Library ieee;
use ieee.std_logic_1164.all;
entity foo is
  port(a, b: in std_logic;
        x, y: out std_logic);
end foo;
```



```
architecture no_buffer_mode of
foo is
  signal inside: std_logic;
begin
  inside <= a AND b;
  x <= inside;
  y <= inside OR (not a);
  -- really wanted y <= x OR (not a);
end no_buffer_mode;
```



## Types



- **We will always use types**
  - **STD\_LOGIC**
  - **STD\_LOGIC\_VECTOR**
  - **These are industry standard and are used when tri-state logic is required.**
  - **These types require the statements**  
USE ieee.std\_logic.ALL;  
LIBRARY ieee;
  - **There are nine STD\_LOGIC types.**
    - U uninitialized
    - X unknown
    - 0 zero
    - 1 one
    - Z tri-state
    - W weak unknown
    - L weak zero
    - H weak one
    - - Don't care



## Designs



- **Designs consist of an ENTITY/ARCHITECTURE pair.**
  - They are usually in the same file. This is a good idea.
  - A file can contain multiple ENTITY/ARCHITECTURE pairs.
    - However one should declare ENTITY/ARCHITECTURE pairs before they are used in another architecture.
  - Altera's MAX+PlusII requires the file name to be xxx.vhd where xxx is the name of an ENTITY in the file.
- **ARCHITECTURE bodies can have two types of statements.**
  - **CONCURRENT**
    - Signal assignment
    - Instantiation
    - When/else
    - With/select
    - Process (as a wrapper for sequential statements)
  - **SEQUENTIAL (only within a process – more later)**
    - Signal assignment
    - If/then/elsif/else
    - Case/when



## Signal Assignment



sum <= ina AND (inb OR inc);

-- One needs parentheses to specify the order.

### ■ Basic Operators

#### □ Unary Arithmetic

- -

#### □ Arithmetic

- +, -, \* What does \* synthesize to?

#### □ Concatenation – defined for strings and signal values

- &

One needs a use clause for the next two – use ieee.std\_logic.all;

#### □ Logical

- AND, NAND, OR, NOR, XOR, XNOR, NOT

#### □ Relational

- =, /=, <, <=, >, >= Note that <= and >= also have other meanings.



## Example – half adder



-- This comment is before the library and use clauses.

library ieee;

use ieee.std\_logic\_1164.all;

-- here is the entity

entity halfadd is

port (a, b : in std\_logic;

sum, c : out std\_logic);

end halfadd;

architecture comp of halfadd is

begin

-- a concurrent statement implementing the and gate

c <= a and b;

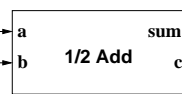
-- a concurrent statement implementing the xor gate

sum <= a xor b;

end comp;

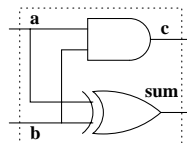
Arithmetic Operation: one bit addition

a	b	sum	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$\text{sum} = a \oplus b$$

$$c = a * b$$





## Example – full adder

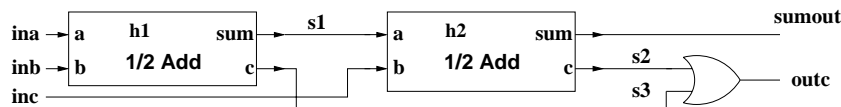


```

library ieee;
use ieee.std_logic_1164.all;
entity fulladd is
  port (ina, inb, inc : in std_logic;
        sumout, outc : out std_logic);
end fulladd;

architecture top of fulladd is
  component halfadd
    port (a, b : in std_logic;
          sum, c : out std_logic);
  end component;
  signal s1, s2, s3 : std_logic;
begin
  -- a structural instantiation of two half adders
  h1: halfadd port map( a => ina, b => inb,
                       sum => s1, c => s3);
  h2: halfadd port map( a => s1, b => inc,
                       sum => sumout, c => s2);
  outc <= s2 or s3;
end top;

```



## Signal Assignments - more



- Consider  $\text{sum} \leq \text{ina} + \text{inb}$ ;
  - All three signals, sum, ina, and inb, must be of the same type.
  - This means that they must be of the same length, e.g.,
    - STD\_LOGIC
    - STD\_LOGIC\_VECTOR(7 DOWNTO 0) -- 0 is the LSB
    - STD\_LOGIC\_VECTOR(0 TO 7) -- 0 is the MSB
  - But an adder has (one) more output bits than input bits.
  - One must say what one means.
    - Do you want to wire a vector backwards?
    - Where does the extra bit come from?



## An Adder

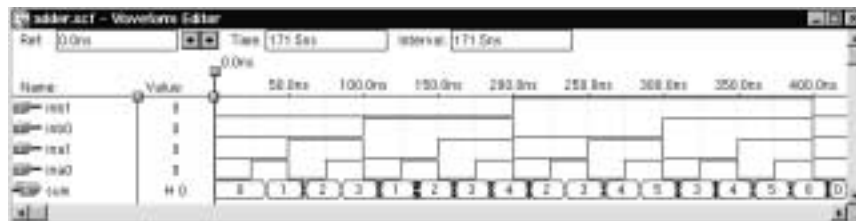


```

library ieee;
use ieee.std_logic_1164.all;
--Use the following for galaxy
--use work.std_arith.all;
--Use the following for MAX+PlusII
use ieee.std_logic_arith.all;
-- Use signed or unsigned as desired.
use ieee.std_logic_signed.all;
--use ieee.std_logic_unsigned.all;

entity adder is
  port (ina: in std_logic_vector(1 downto 0);
        inb: in std_logic_vector(1 downto 0);
        sum: out std_logic_vector(2 downto 0));
end adder;
architecture behavioral of adder is
begin
  sum <= ('0' & ina) + ('0' & inb);
end behavioral;

```



## Bad Adder – functionally incorrect

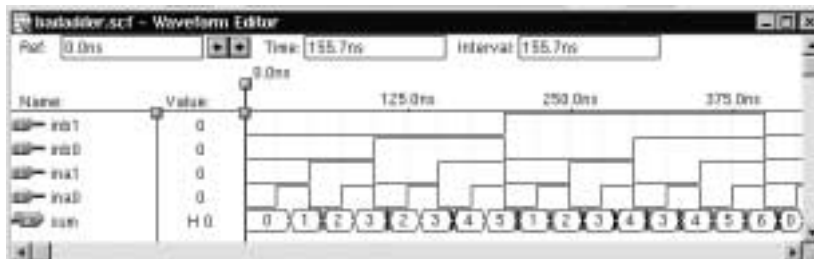


```

library ieee;
use ieee.std_logic_1164.all;
--Use the following for galaxy
--use work.std_arith.all;
--Use the following for MAX+PlusII
use ieee.std_logic_arith.all;
-- Use signed or unsigned as desired.
use ieee.std_logic_signed.all;
--use ieee.std_logic_unsigned.all;

entity badadder is
  port (ina: in std_logic_vector(1 downto 0);
        inb: in std_logic_vector(0 to 1);
        sum: out std_logic_vector(2 downto 0));
end badadder;
architecture behavioral of badadder is
begin
  sum <= ('0' & ina) + ('0' & inb);
end behavioral;

```





## Instantiation



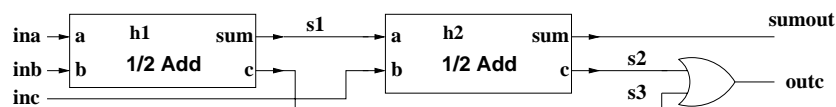
### ■ Instantiation by named association

```
h1: halfadd port map( a => ina, b => inb,
                    sum => s1, c => s3);
```

### ■ Instantiation by positional association (not recommended)

□ It is too easy to wire to the wrong ports.

```
h1: halfadd port map(ina, inb, s1, s3);
```



## When - Else



### ■ When - else assigns a signal a value when a condition is true.

- The conditions need not be mutually exclusive.
  - The first one that is true “wins”.
- You must have an ELSE at the end so at least one condition is true,

```
signal <= val1 WHEN conda ELSE
        val2 WHEN condb ELSE
        val3 WHEN condc ELSE
        ...
        val4;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity priority is port (
    a, b, x, y: in std_logic;
    j: out std_logic);
end priority;
```

```
architecture logic of priority is
begin
    j <= x when a='1' else
        y when b='1' else
        '0';
end logic;
```



## Example – when/else



```

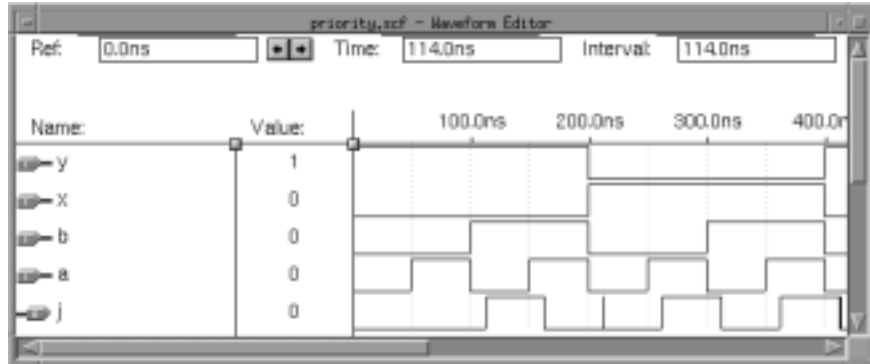
library ieee;
use ieee.std_logic_1164.all;
entity priority is port (
  a, b, x, y: in std_logic;
  j: out std_logic);
end priority;

```

```

architecture logic of priority is
begin
  j <= x when a='1' else
    y when b='1' else
    '0';
end logic;

```



## With – Select - When



- **With – select – when makes a signal assignment based on the value of a selection signal.**
  - The **WHEN** clauses must be mutually exclusive.
  - All signal values must be specified.
    - This is hard (impossible) to do when using `std_logic`.
    - Always use a **WHEN OTHERS**; at the end.

```

WITH sel_sig SELECT
sig_name <= val1 WHEN vala_of_sel_sig,
           val2 WHEN valb_of_sel_sig,
           ...
           val3 WHEN OTHERS;

```

```

library ieee;
use ieee.std_logic_1164.all;
entity mux is port (
  a, b, c, d: in std_logic_vector(4 downto 0);
  s: in std_logic_vector(1 downto 0);
  x: out std_logic_vector(4 downto 0));
end mux;

```

```

architecture archmux of mux is
begin
  with s select
    x <= a when "00",
        b when "01",
        c when "10",
        d when others;
end archmux;

```



## Example – with/select/when



```
library ieee;  
use ieee.std_logic_1164.all;  
entity mux2 is port (  
  a, b: in std_logic;  
  s: in std_logic;  
  x: out std_logic);  
end mux2;
```

```
architecture archmux of mux2 is  
begin  
  with s select  
    x <= a when '0',  
      b when others;  
end archmux;
```

