



L3: Some on Lab 1 and VHDL for Combinational Logic



Some slides are derived from slides used in past terms of 6.111



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;

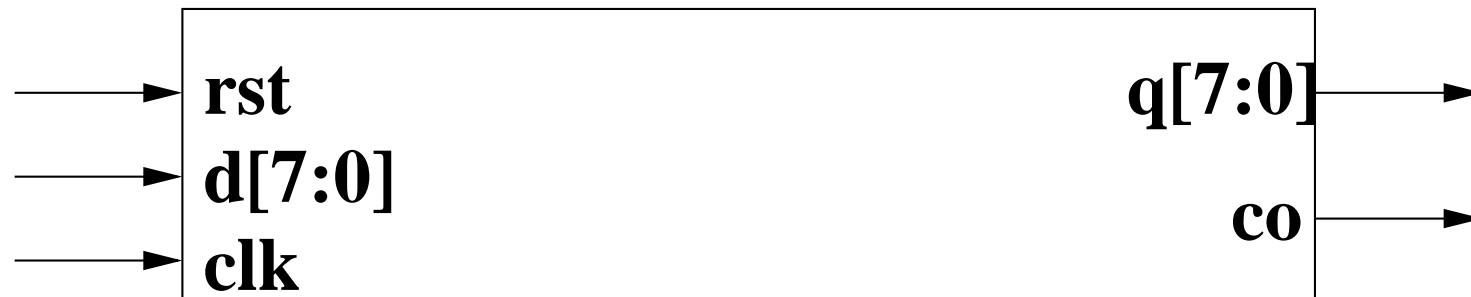
 d : in STD_LOGIC_VECTOR(7 downto 0);

 q : out STD_LOGIC_VECTOR(7 downto 0);

 co : out STD_LOGIC);

end black_box;

black_box





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



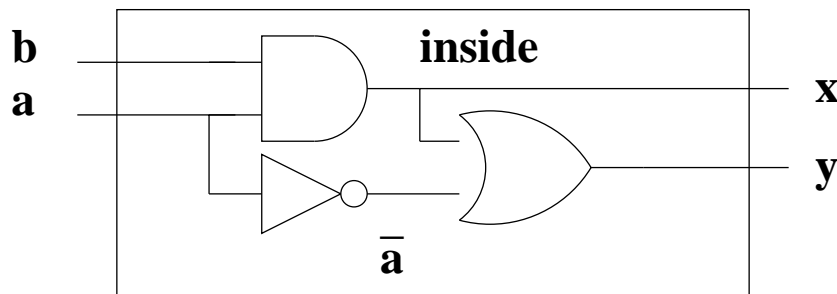
Not Needing Mode BUFFER



■ 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;
```



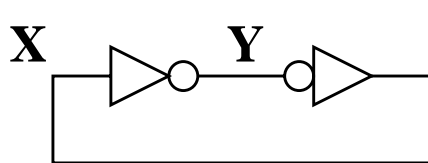
```
architecture no_buffer_mode of foo is  
  signal inside: std_logic;  
begin  
  inside <= a AND b;  
  x <= inside;  
  y <= inside OR (not a);  
  -- really wanted y <= x OR (not a);  
end no_buffer_mode;
```



Feedback

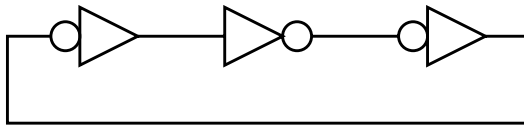


Feedback produces 'State' which can be used to store information.



X	Y
0	1
1	0

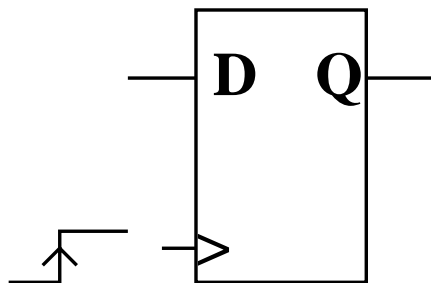
Note that either state (X=0, Y=1 or X=1, Y=0) is valid.



?

Try this in the lab.

What does this one do? 



D	Q _n
0	0
1	1

Flip-flops are used to store state.

D Flip-Flop



Extract of Lab 1



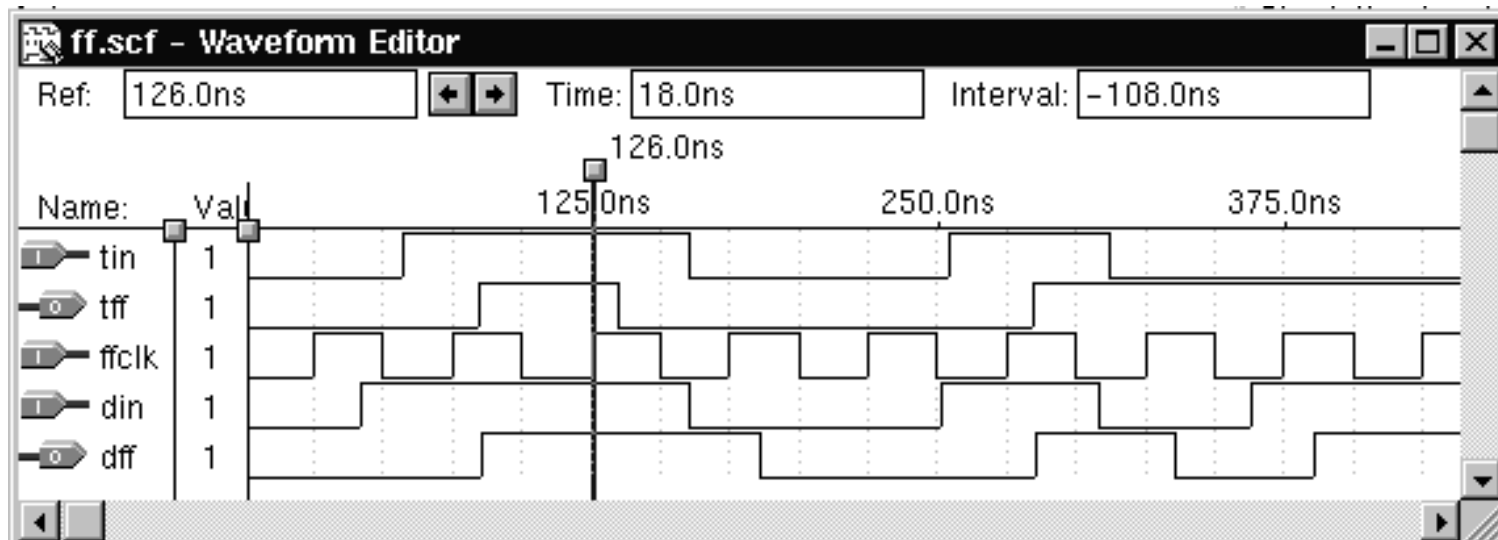
```
library ieee;
use ieee.std_logic_1164.all;
entity ff is
  port (ffclk, din, tin, jin, kin : in std_logic;
        dff, tff, jkff : out std_logic);
end ff;
architecture behavioral of ff is
  signal insidetff, insidejkff : std_logic;
begin
  process (ffclk)
  begin
    if rising_edge(ffclk) then
      dff <= din;
      insidetff <= tin xor insidetff;
      insidejkff <= (jin and (not insidejkff)) or ((not kin) and insidejkff);
    end if;
  end process;
  tff <= insidetff;
  jkff <= insidejkff;
end behavioral;
```



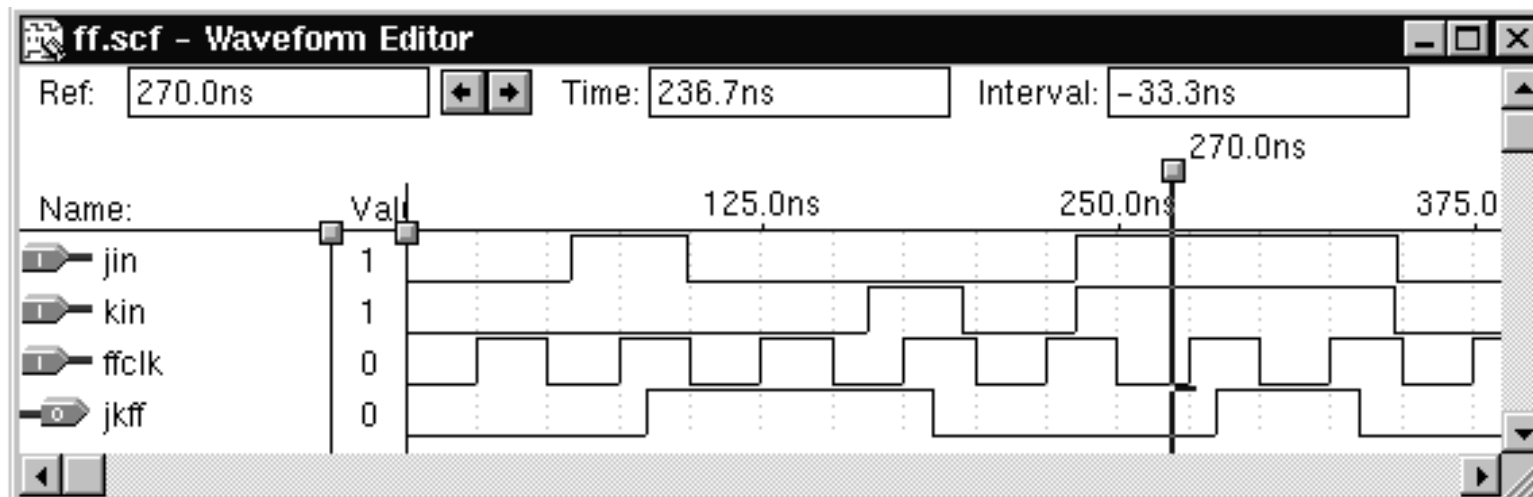
Simulation of the Extract



dff <= din; **tff <= tin xor tff;**



jkff <= (jin and (not jkff)) or ((not kin) and jkff);





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



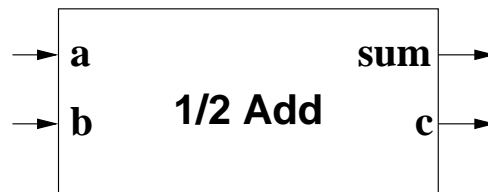
```

-- 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;

```

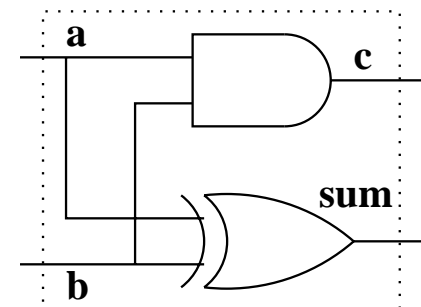
Arithmetic Operation: one bit addition

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



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

$$c = a * b$$





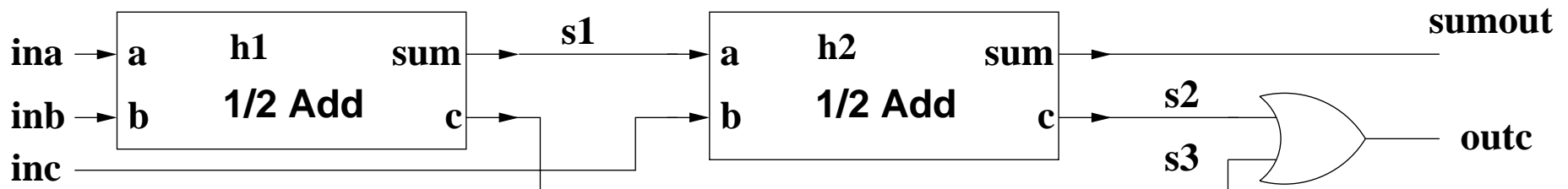
Example – full adder



```
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 DOWNT0 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?

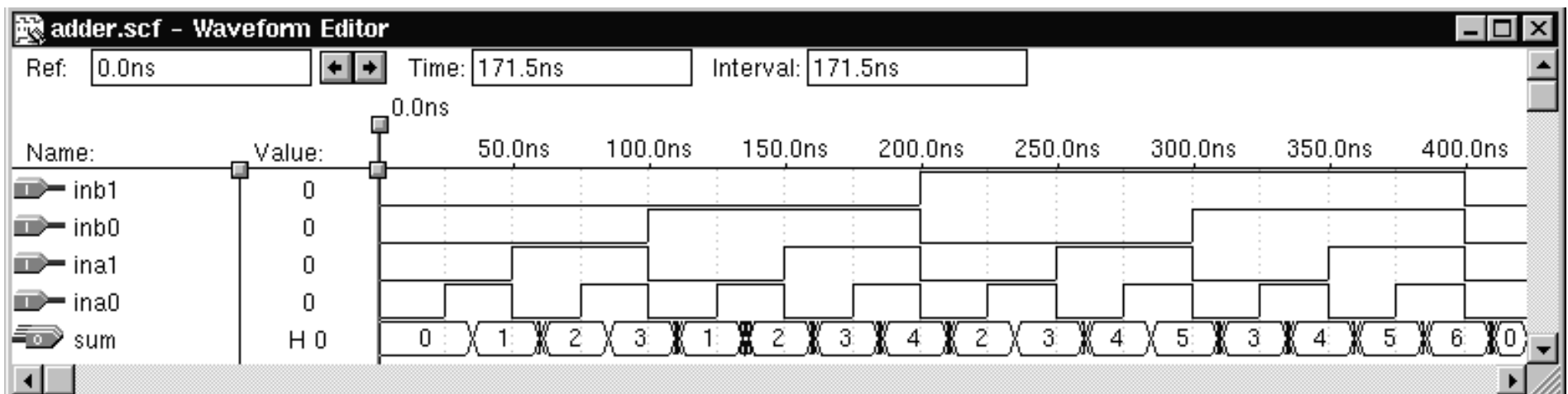


An Adder



```
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;
```



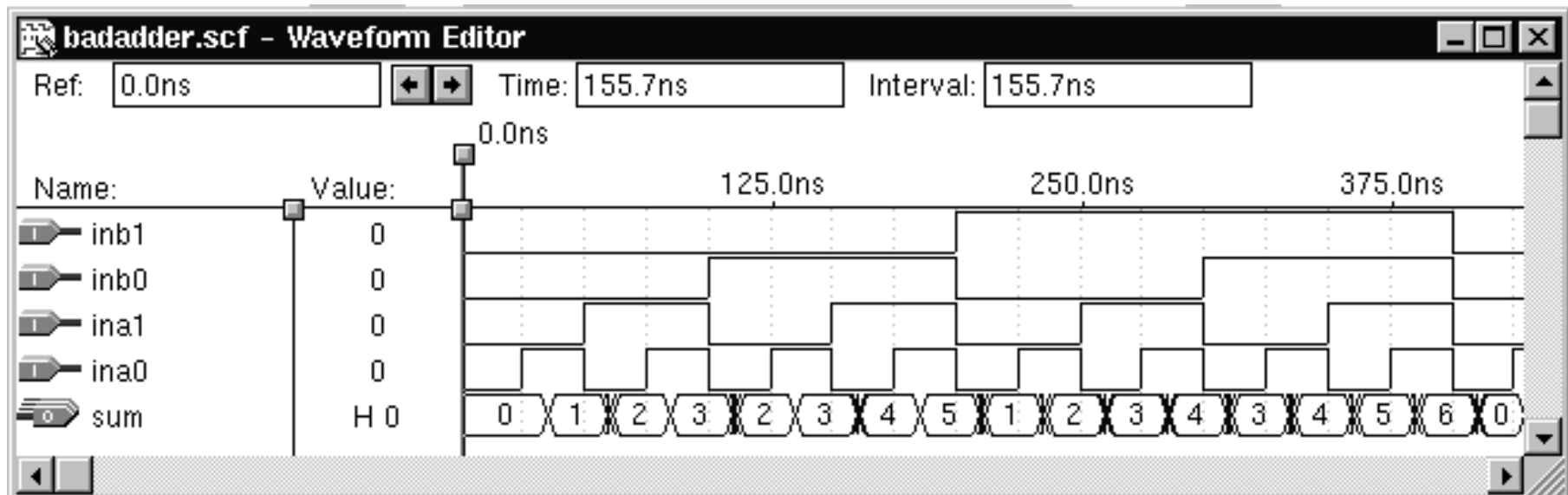


Bad Adder – functionally incorrect



```
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;
```





Review - Signal Assignment/Instantiation



- Signal Assignment

- `sum <= ina AND (inb OR inc);`

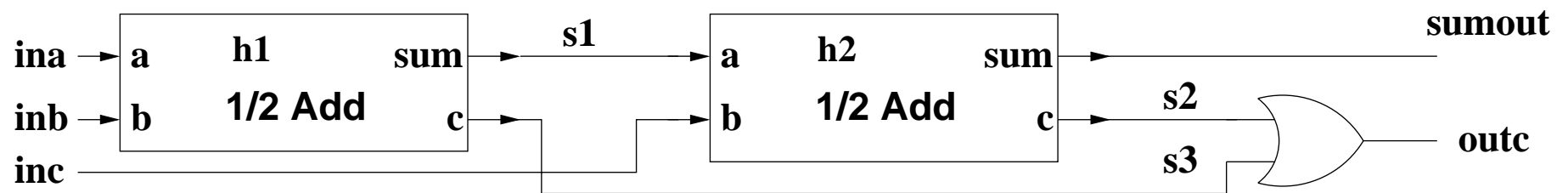
- Instantiation by named association

- `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;
```

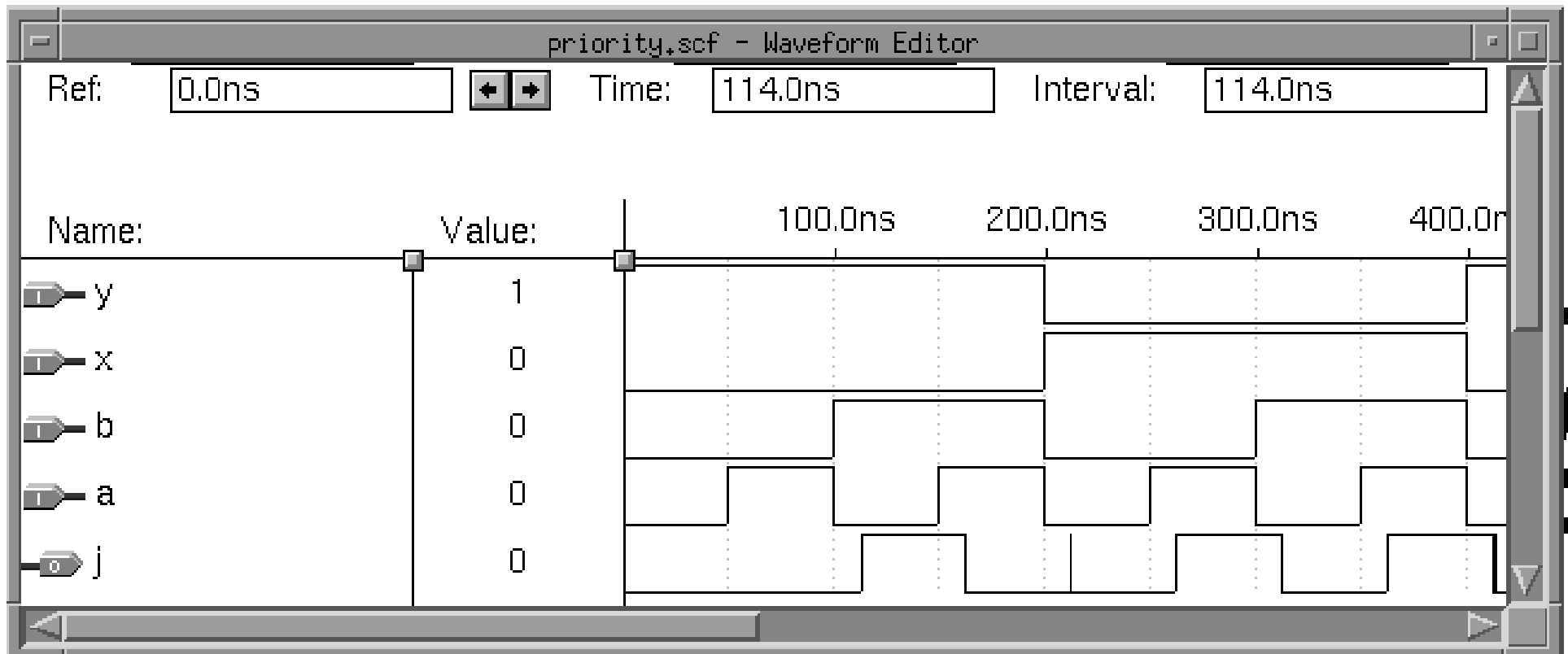


Example – when/else



```
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;
```

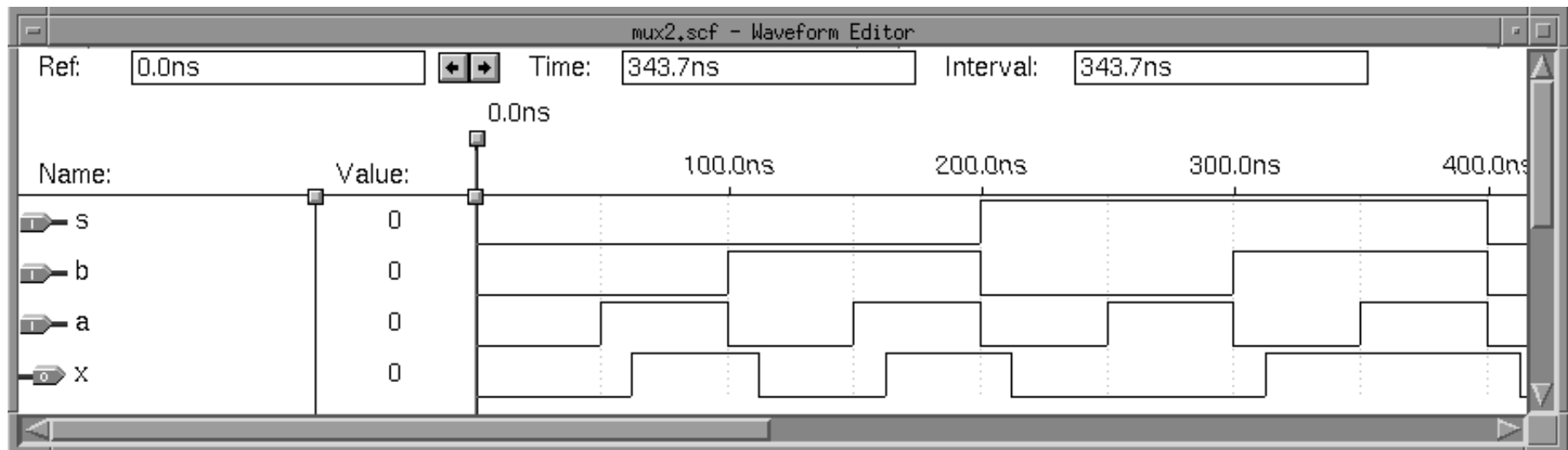


Example – with/select/when



```
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;
```





Negative True and VHDL



- **Signals in VHDL are inherently positive true.**
 - A signal is high (a one) when it “happens”.
- **A negative true signal is low (a zero) when it “happens”.**
 - Negative true signals are usually only used on I/O.
 - It is nice to be able to recognize whether a signal is negative or positive true from a clue provided by the signal name.
 - /foo is out as a VHDL identifier cannot begin with a slash.
 - nfoo could be troublesome for signal names starting with n.
 - not_foo and neg_true_foo are too verbose.
- **By convention, we will prepend n_ to all signal names when the signal is negative true (low when it ‘happens’).**

N_foo



Negative or Positive True?



```
library ieee;
use ieee.std_logic_1164.all;
entity neg is
  port (a1, b1, a2, b2, a3, b3 : in std_logic;
        x, n_y, n_z : out std_logic);
end neg;
architecture equations of neg is
  signal y : std_logic;
begin
  -- The only clue we have are the signal names.
  -- The next two are positive true.
  x <= a1 AND b1;
  y <= a2 AND b2;
  -- The next two are negative true.
  n_y <= not y;
  n_z <= not (a3 OR b3);
end equations;
```