CPE 487: Digital System Design

Spring 2018

Lecture 5

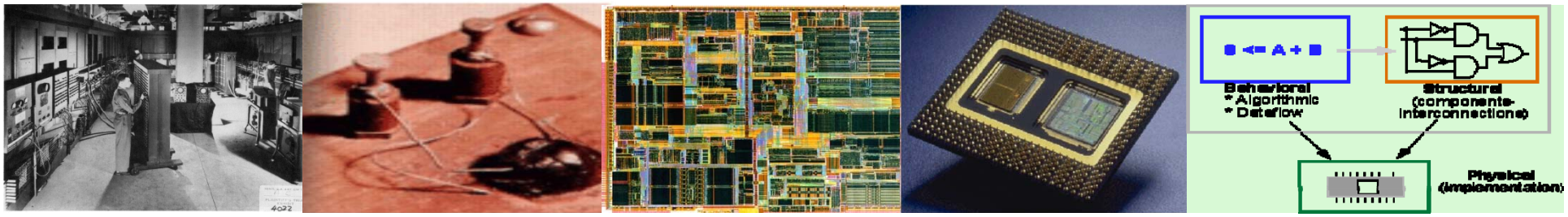
Dataflow Modeling

Bryan Ackland

Department of Electrical and Computer Engineering

Stevens Institute of Technology

Hoboken, NJ 07030



Concurrent Signal Assignment

- Simple CSA's execute whenever an event occurs on a signal on the RHS of the assignment statement

  z <= (a **and** (**not** b)) **xor** (c **or** (a **and** (**not d**))) **after** 3 **ns;**

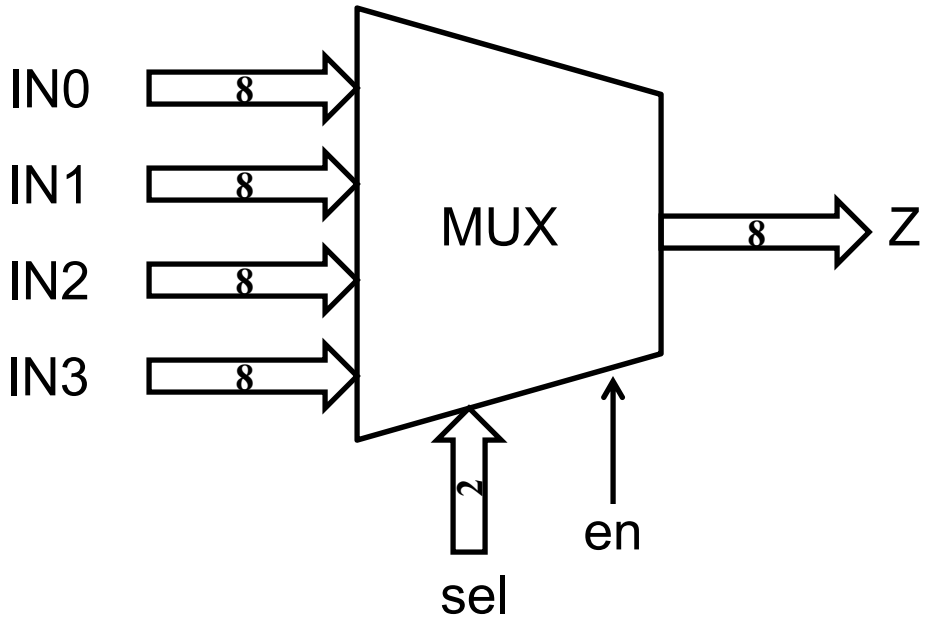
  target-signal <= *waveform-elements*;

- Useful for describing gate level combinational logic
  - (when output is a function of current input values only)

- Not suited for modeling at higher levels of abstraction

- Do not capture sequential logic behavior
  - (when output is a function of current & previous input values)

4-way 8-bit Multiplexer



- Using single bit assignment statements, this would require 54 assignment statements!
- Even using 8-bit vectors (std_logic_vector) would require 12 statements
- Simplify using conditional expressions

Conditional Signal Assignment Statement

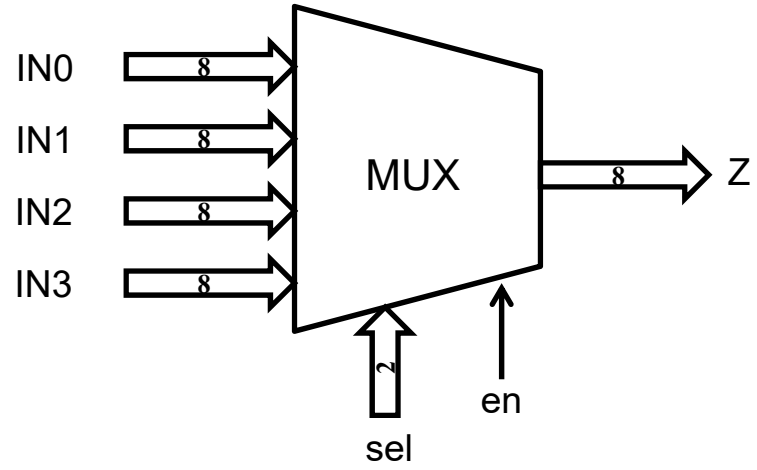
- Conditional signal assignment statement selects different values for the target signal based on the various specified conditions – it is like an if-then-else statement.

target-signal <= [*waveform-elements* **when** *condition* **else**]
[*waveform-elements* **when** *condition* **else**]
…
[*waveform-elements* **when** *condition* **else**]
[*waveform-elements*];

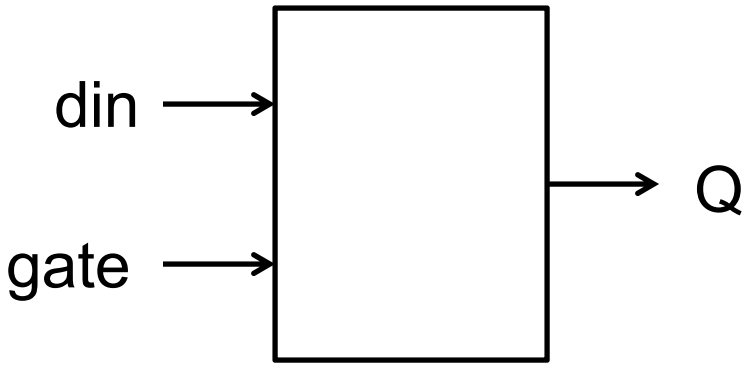
- Will be executed whenever an event occurs on a signal used in any of the waveform expressions, or in any of the conditions.
- Only the first clause found to be true is executed
  - Order of these clauses matters!

4-way 8-bit Multiplexer

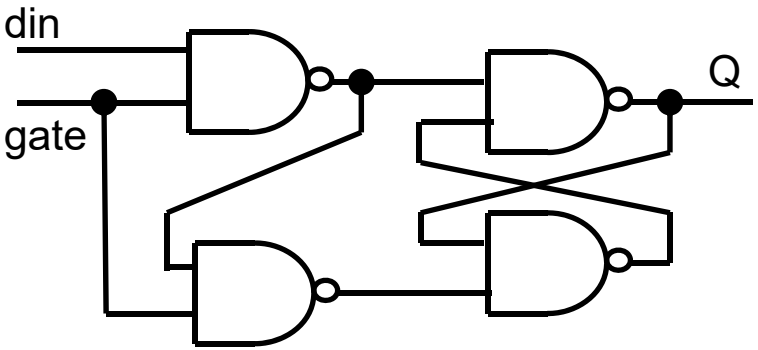
```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4 is
        port ( IN0, IN1, IN2, IN3: in std_logic_vector (7 downto 0);
        sel: in std_logic_vector (1 downto 0);
        en: in std_logic;
        Z: out std_logic_vector (7 downto 0));
end entity mux4;

architecture behavioral of mux4 is
begin
        Z <= "00000000" after 5 ns when en='0' else
                IN0 after 5 ns when sel="00" else
                IN1 after 5 ns when sel="01" else
                IN2 after 5 ns when sel="10" else
                IN3 after 5 ns when sel="11" else
                "XXXXXXXX" after 5 ns;
end architecture behavioral;
```



1-bit Latch



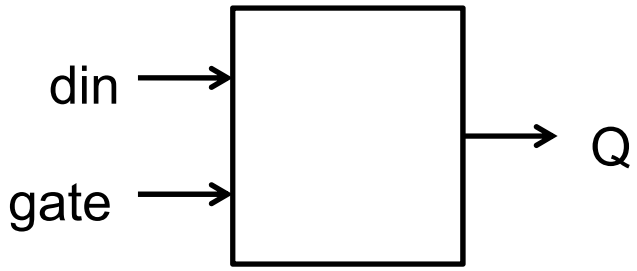
din	gate	Q
0	1	0
1	1	1
0	0	*no change*
1	0	*no change*



- Sequential circuit

- Gate level structure describes a possible implementation but does not explicitly capture behavior

1-bit Latch using Conditional Assignment




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

entity latch is
      port (din, gate: in std_logic;
      Q: out std_logic);
end entity latch;

architecture lat_1 of latch is
begin
      Q <= din after 3 ns when gate='1';
end architecture lat_l;
```

- There are combinations of inputs which will not trigger execution ⇨ memory of previous state

Selected Signal Assignment Statement

- Alternative form of conditional assignment
- Selects different values for a target signal based on the value of a select expression.
  - more like a case statement.

with *expression* **select**

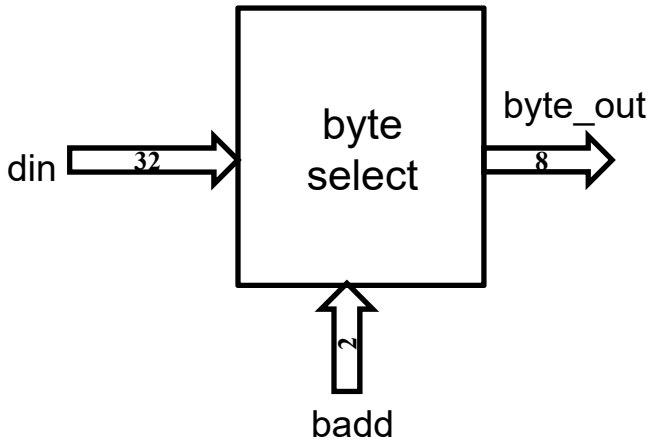
target-signal <= *waveform-elements* **when** *choices*,
waveform-elements **when** *choices*,
…
[*waveform-elements* **when others**];

- Executed when event occurs on any signal in the select expression or on any signal used in any waveform expression.
- Choices must be mutually exclusive
  - all choices are evaluated – order does not matter

Select Signal Assignment: Byte Selector

- Select a specified byte from 32-bit word:

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

entity bytesel is
        port ( din: in std_logic_vector (31 downto 0);
        badd: in std_logic_vector (1 downto 0);
        byte_out: out std_logic_vector (7 downto 0));
end entity bytesel;
```




```
architecture dataflow of  bytesel is
begin
        with badd select
                byte_out <= din(7 downto 0) after 3 ns when “00”,
                        din(15 downto 8) after 3 ns when “01”,
                        din(23 downto 16) after 3 ns when “10”,
                        din(31 downto 24) after 3 ns when “11”,
                        “XXXXXXXX” after 3 ns when others;
end architecture dataflow;
```

Sidebar: Concatenation Operator

- & operator can be used to concatenate a j-bit word and a k-bit word to produce a (j+k) bit word.

- For example:

if a= "0010" and b="1010" and of type std_logic_vector(3 downto 0):

a & b = "00101010"

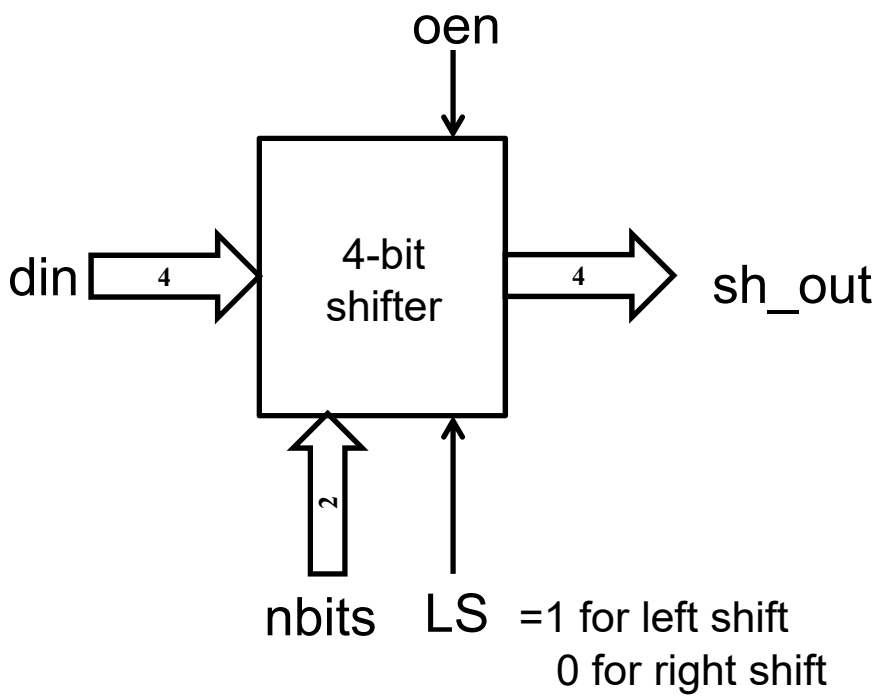
a & b & a = "001010100010"

a(3 downto 2) & "01" =

b(3) & a(1 downto 0) =

Example: 4-bit Logical Shifter

- Use Conditional Signal Assignment to describe a circuit that logically shifts a 4-bit word 0-3 bits to the right or left. Include tri-state output.

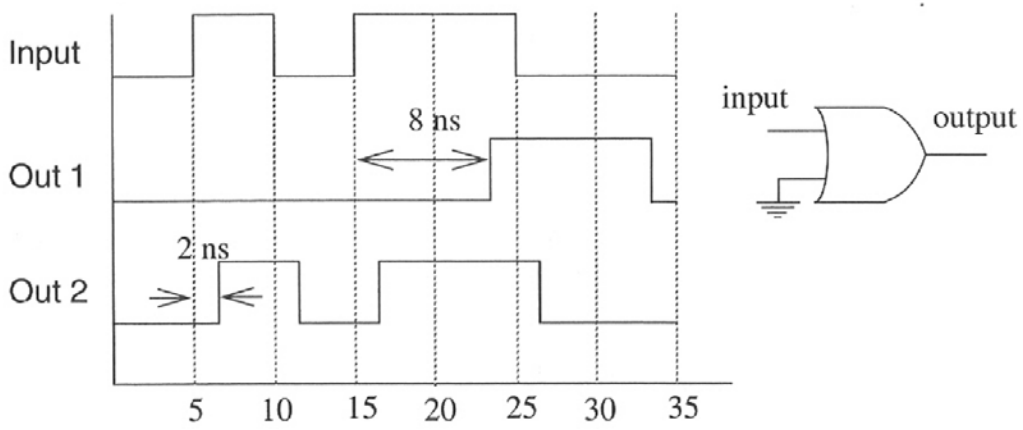


Inertial Delay

- By default, gate delays are considered inertial
  - due to intrinsic speed limitation of device
- What happens if a gate sees a very short pulse at its input?
- Inertial delay will filter out pulses shorter than the specified gate delay
  - Default rejection window = gate delay

out1 <= input **or** '0' **after** 8 ns.

out2 <= input **or** '0' **after** 2 ns.



Inertial Delay with Rejection Window

- We can override default rejection window:

  z <= **reject** 3ns **inertial** (x xor y) **after** 5ns;

- This gate has a delay of 5ns, but will only reject input pulses shorter than 3 ns.
- Inertial delay will also reject output pulses shorter than the reject window.
- For example, using xor gate (above) with following inputs:

  x <= '0', '1' **after** 10ns, '0' **after** 20ns;

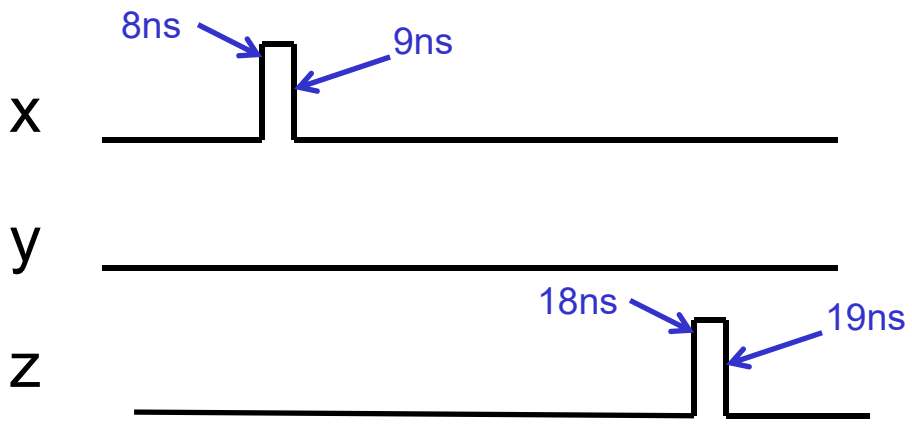
  y <= '0', '1' **after** 12ns, '0' **after** 18ns;

  generates no output on z

Transport Delay

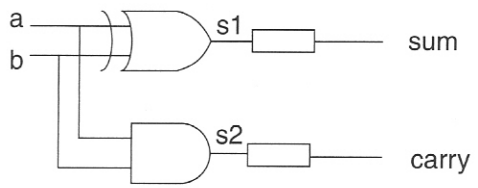
- Signals also experience delays through wires
  - Wires can change state very quickly (fast rise & fall time)
  - If wire is long, transport delay can be much greater than the wire rise/fall time.
  - Even a very long wire, may transmit very short pulses (with an appropriate delay
- Transport delay does not filter out short pulses:

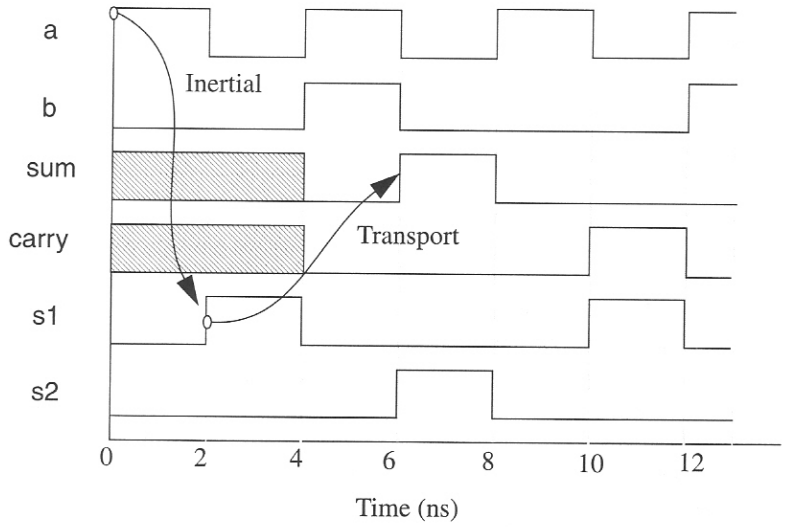
```
sum <= transport (x xor y) after 10ns;
```



Inertial + Transport Delay

```
library IEEE;
use IEEE.std_logic_1164.all;
entity half_adder is
port(a, b: in std_logic;
       sum, carry: out std_logic);
end entity half_adder;

architecture transport_delay of half_adder is
signal s1, s2: std_logic:= '0';
begin
s1 <= (a xor b) after 2 ns;
s2 <= (a and b) after 2 ns;
sum <= transport s1 after 4 ns;
carry <= transport s2 after 4 ns;
end architecture transport_delay;
```



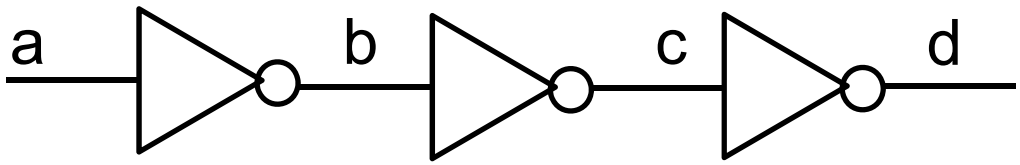


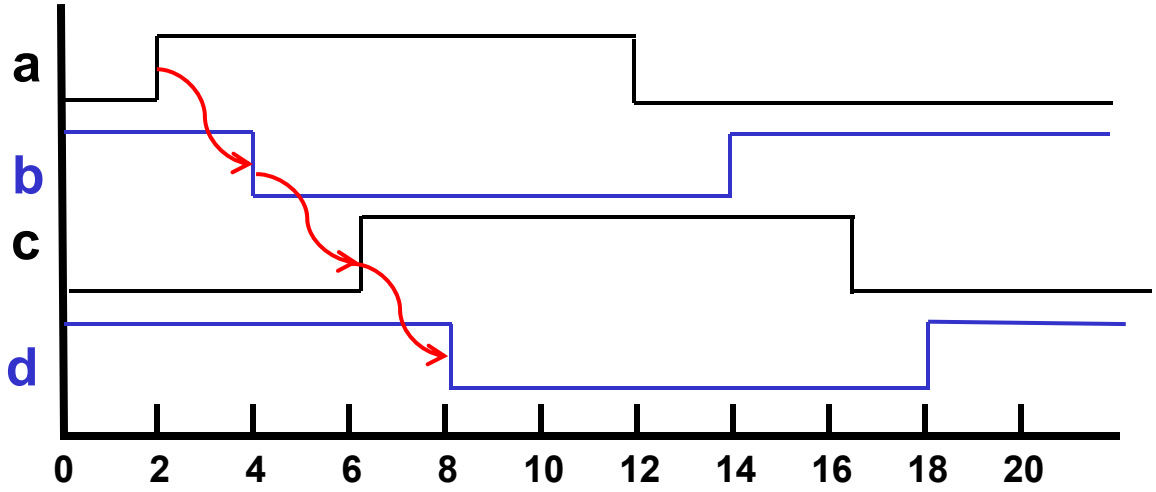
- Allows us to independently model gate (inertial) and wire (transport) delays

Dataflow Delays

- Dataflow describes process of events flowing from one device to another

```
b <= not a after 2 ns;
c <= not b after 2 ns;
d <= not c after 2 ns;
```

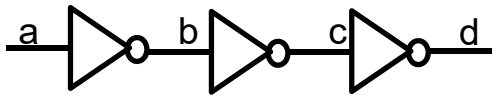




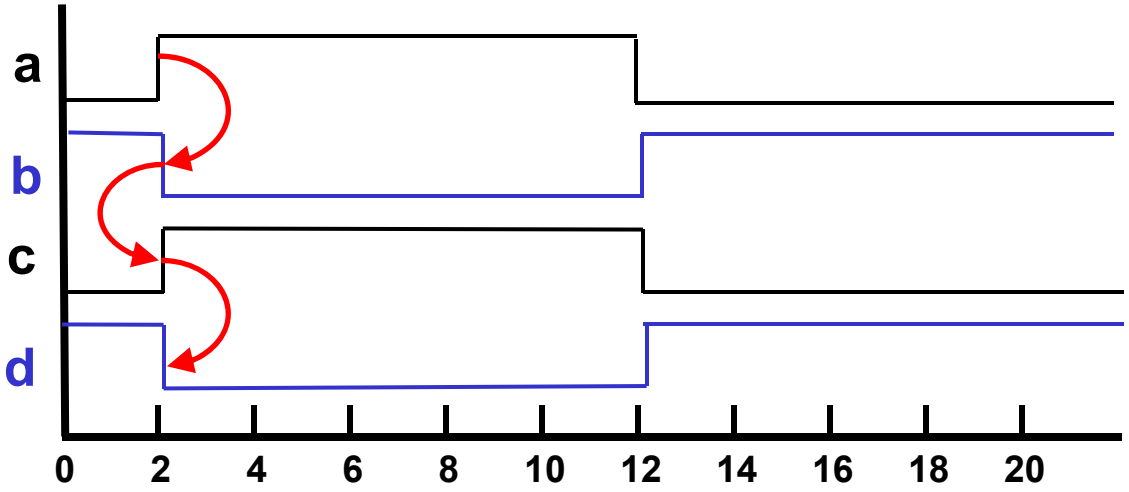
Zero Delays

- What happens when delays are zero?

e.g.: b <= **not** a **after** 0 ns; *or* b <= **not** a;



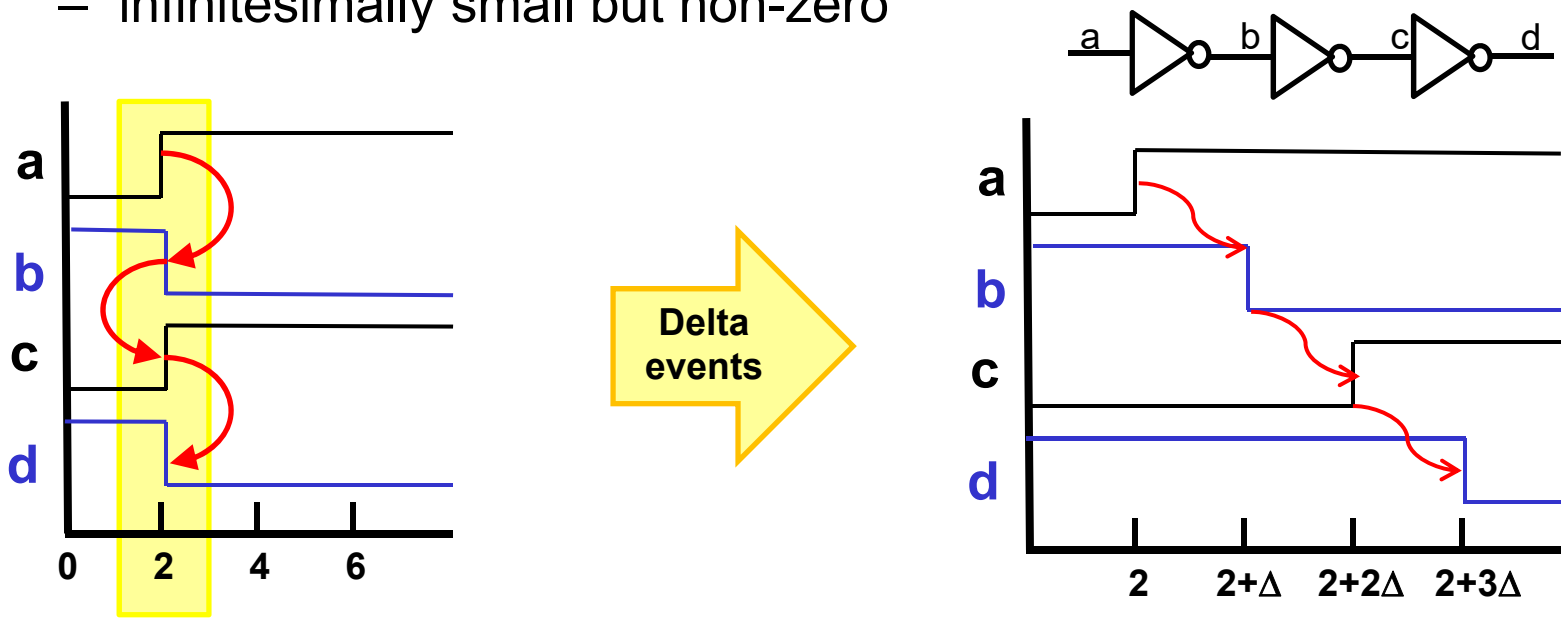

```
c <= not b;
d <= not c;
```



- Are these events really happening in zero time?
- What impact does this have on concurrency?

Delta Delays

- When no delay is specified, simulator adds small (delta) delay Δ when scheduling the output event
- Δ is smaller than any physical delay
  - infinitesimally small but non-zero



- This maintains correct data flow and ensures events processed in correct order
  - without introducing physical delay

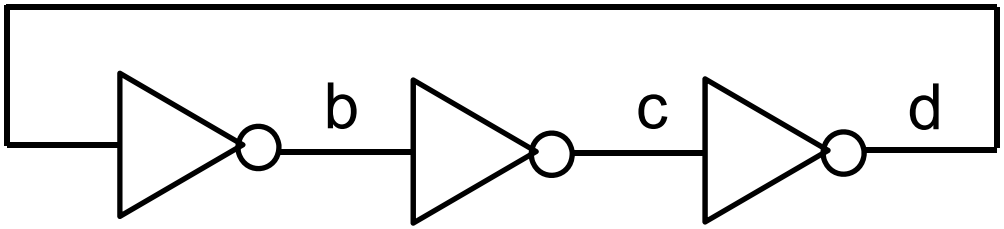
Signal Delays - Summary

- VHDL signals do not change instantaneously.
- A scheduled change for a VHDL signal never occurs at the present time but is always delayed until some future time.
  - this is the basis of concurrency
  - execution of one CSA cannot affect the execution of another CSA *at the present time*
- The future time at which the change is to take affect can be explicitly stated. If no time is specified for a signal change, the default future time is the present time plus an infinitesimally small time called delta time.

Example: Ring Oscillator

- What will happen if the output of the inverter chain is fed back to the input?

```
b <= not d after 2 ns;
c <= not b after 2 ns;
d <= not c after 2 ns;
```

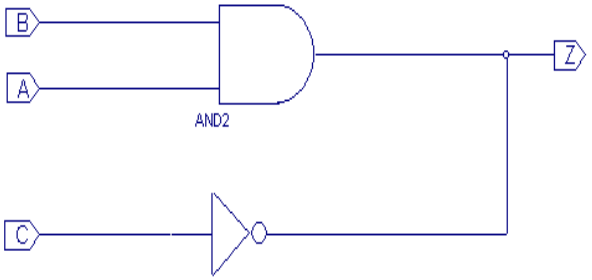


- What will happen if no delay is specified?

Multiple Drivers

- Each concurrent signal assignment statement creates a driver for the signal being assigned
- Can there be more that one driver for a signal?
  - depends on the type of the signal

```
z <= a and b after 10 ns;
z <= not c after 5 ns;
```

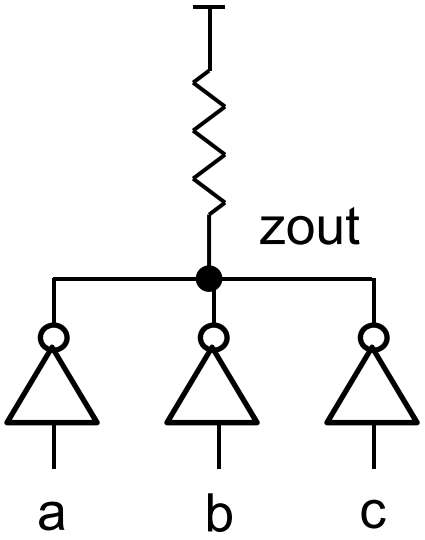


- With standard (unresolved) types (e.g. bit, std_ulogic, integer), this is illegal and will cause either a compiler or a run-time error
- With resolved types (e.g. std_logic, std_logic_vector) a resolution function is invoked to determine correct result (more on this later).

Std_logic Resolution Table

	U	X	0	1	Z	W	L	H	-
uninitialized **U**	U	U	U	U	U	U	U	U	U
unknown **X**	U	X	X	X	X	X	X	X	X
forcing ‘0’ **0**	U	X	0	X	0	0	0	0	X
forcing ‘1’ **1**	U	X	X	1	1	1	1	1	X
high impedance **Z**	U	X	0	1	Z	W	L	H	X
weak unknown **W**	U	X	0	1	W	W	W	W	X
weak ‘0’ **L**	U	X	0	1	L	W	L	W	X
weak ‘1’ **H**	U	X	0	1	H	W	W	H	X
don’t care **-**	U	X	X	X	X	X	X	X	X

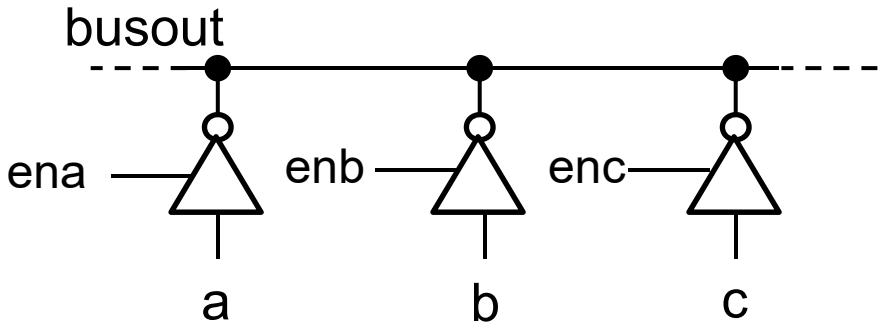
Multiple Driver Examples



"wired NOR"

signal zout has 4 drivers
(3 open-drain buffers plus a resistor)

buffers output '0' or 'Z'
resistor outputs 'H'



tri-state bus

each buffer outputs '0', '1 or 'Z'
(only one driver active at a time)

Concurrent Assertion Statement

- During simulation and debugging it is useful to be able to check and report on signal values, e.g:
  - Illegal combination of inputs
  - setup or hold time violations
  - unexpected condition
- Assert statement provides mechanism for testing state of system and reporting results on simulator console

  assert *boolean-expression*
  [**report** *string-expression*]
  [**severity** *expression*];

- If the value of the *boolean-expression* is false, the report message is printed along with the severity level
  - executed when event occurs on any signal in *boolean-expression*

Severity Levels

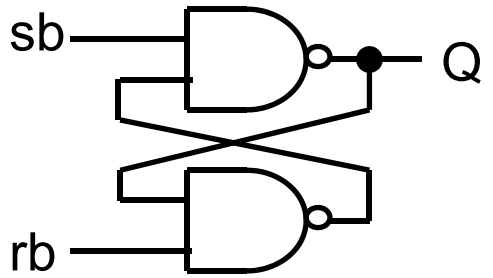
- Implementation dependent
- Common values:
  - NOTE
  - WARNING
  - ERROR
  - FAILURE *(this level will abort Xilinx Isim simulator)*

for example:

assert (a=b) **or** (a=c)
[**report** "a is not equal to b or c"]
[**severity** WARNING];

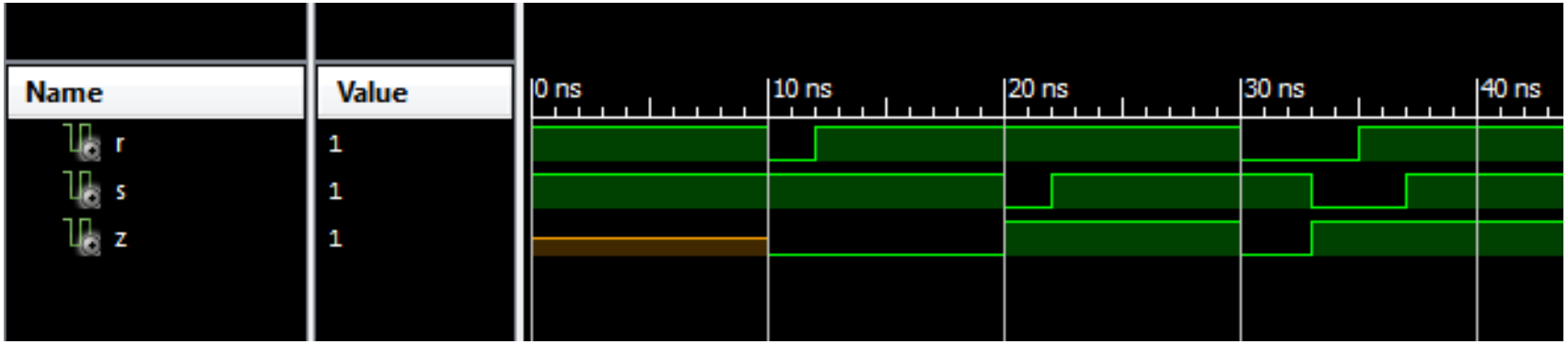
Example: RS Latch

```
entity rsff is
port(rb, sb: in std_logic;
    Q:out std_logic);
end entity rsff;

architecture rsa1 of rsff is
begin
    assert rb='1'
    report  "reset initiated"
    severity NOTE;

    assert (rb='1') or (sb='1')
    report  "rb and sb both zero"
    severity ERROR;

    Q <= '1' when sb='0' else
        '0' when rb='0';

end architecture rsa1;
```



rb	sb	Q
0	1	0
1	0	1
1	1	*no change*
0	0	*illegal*

RS Latch: Simulation Output



Console

```
# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 10 ns: Note: reset initiated (/rs_assert_tb/uut/).
at 30 ns: Note: reset initiated (/rs_assert_tb/uut/).
at 33 ns: Error: Both r and s are zero
ISim>
```