Delay Model

- Two Delay Mechanisms
  - Transport
  - Inertial
  - Delta delay (where no delays are specified)

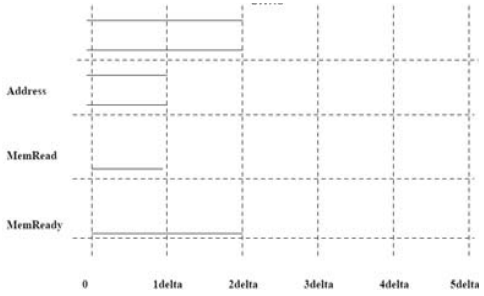
```
architecture abstract of
ComputerSystem is
subtype word is bit vector(31 downto 0);
signal address: natural;
signal readdata, writedata :word;
signal MemRead, MemWrite,
MemReady : bit := '0';
begin
```

```
cpu : process is
variable instLreg : word;
variable PC : natural;
begin
loop
        address <= PC; MemRead <= '1';
        wait until MemReady = '1';
        instrreg := readdata; MemRead <= '0';
        wait until MemReady = '0';
        PC := PC + 4;
        -- execute the instruction
end loop;
end process cpu;
```

```
memory : process is
type memoryarray is array (0 to 2**14 - 1) of word;
variable store : memoryarray
begin
        wait until MemRead = '1' or MemWrite = '1';
        if MemRead = 'l'then
                readdata <= store(address/4); MemReady <= '1';
                wait until MemRead = '0'
                MemReady <= '0';
        else -- perform write access
        end if;
end process memory;
end architecture abstract;
```

Delta Delay

```
    loop
        address <= PC; MemRead <= '1';
        wait until MemReady = '1';
        instrreg := readdata; MemRead <= '0';
        wait until MemReady = '0';
        PC := PC + 4;
        -- execute the instruction
    end loop;
```




```
begin
    wait until MemRead = '1' or MemWrite = '1';
    if MemRead = 'l'then
        readdata <= store(address/4); MemReady <= '1';
        wait until MemRead = '0'
        MemReady <= '0';
    else -- perform write access
    end if;
```

VHDL Basic Modeling Constructs

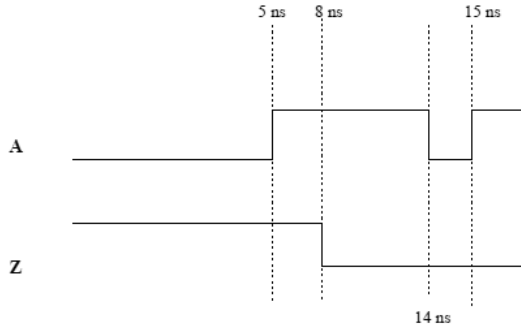
VLSI CAD
Case Western Reserve University

Dan Saab

Inertial

- This assignment rejects pulses shorter than 3 ns

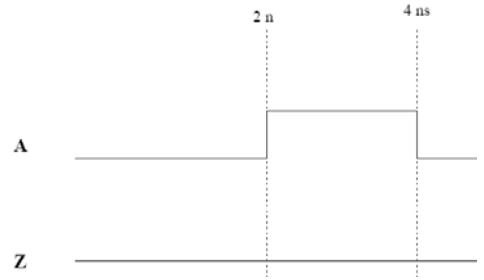
```
aexmp: process (a) is
begin
      Z <= inertial not a after 3 ns;
end process aexmp;
```



Transport

- transaction may delete already scheduled transaction when the new transaction is scheduled at earlier time.

```
aexmp: process (a) is
constant rise: time 8 ns;
constant fall: time 5 ns;
begin
      if a = '1' then
            z <= transport a after rise;
      else -- a='0'
            z <= transport a after fall;
      end if
end process aexmp;
```



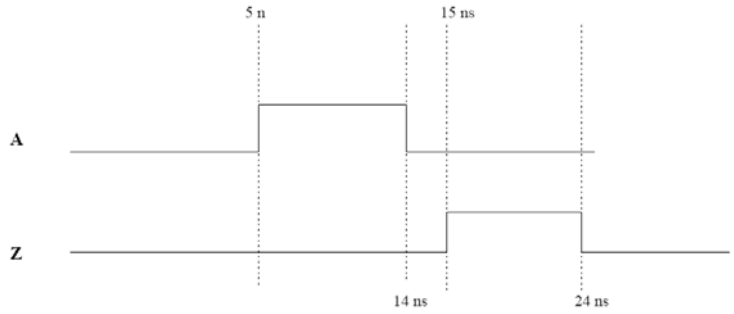
Inertial Delay

- When an gate input changes, the output tends to stay in the same state unless then input is applied for a sufficiently long duration.
  - VHDL uses inertial delay mechanism to model this behavior
    - Reject input pulses of short duration
- Inertial delay is the mechanism used by default in a signal assignment.

Transport

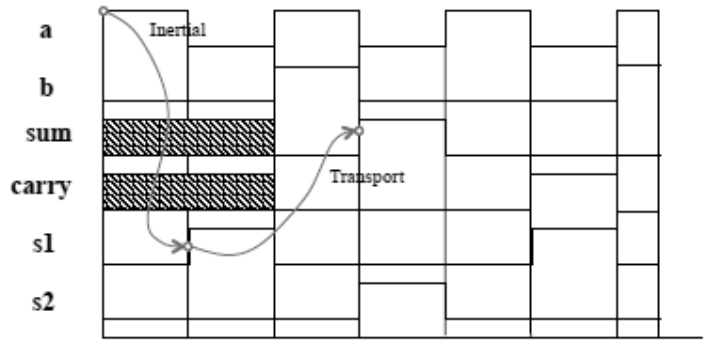
- All input events are propagated to output signals

```
dexmp: process (A) is
begin
    z <= transport A after 10 ns;
end process dexmp;
```



Transport/Inertial

```
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;
```



Inertial

- Rejection can be specified

```
aexmp: process (a) is
begin
    Z <= reject 2 ns inertial not a after 3 ns;
end process aexmp;
```

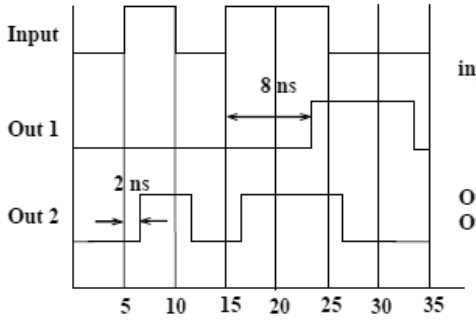
Inertial

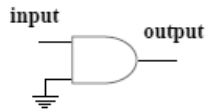
```
library ieee; use ieee.std logic 1164.all;
entity someckt is
port ( a, b: in std ulogic; zout : out std_ulogic);
end entity someckt;
architecture exmp1 of someckt is
signal res: std_ulogic;
begin
      gate : process (a, b) is
      begin
            res <= a and b;
      end process gate;

      delay: process (res) is
      begin
            if res = '1' then
                  zout <= reject 4 ns inertial '1' after 6 ns;
            elsif res = '0' then
                  zout <= reject 3 ns inertial '0' after 5 ns;
            else
                  zout <= reject 2 ns inertial 'X' after 4 ns;
            end if;
      end process delay;
end architecture exmpl;
```

Inertial Delays

- Out1 <= inertial Input after 8 ns;
- Out2 <= inertial Input after 2 ns;



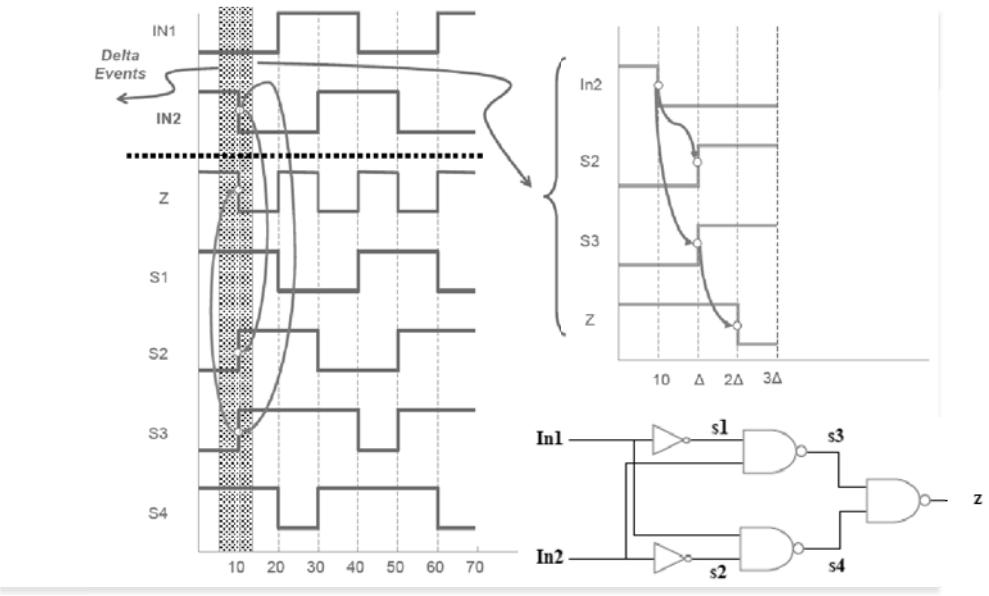


Out 1: gate propagation delay 8ns
Out 2: gate propagation delay 2ns

Conditional Signal Assignment Statements

- Conditions may overlap, as for the if statement. The expression corresponding to the first "true" condition is assigned.

```
architecture COND of BRANCH is
begin
  Z <= A when X = 5 else
       B when X < 10 else
       C;
end COND;
```



Conditional Signal Assignment Statements

- Syntax: Each condition is aboolean expression

```
name <= delaymechanism
            waveform when cond1 else
            waveform when cond1 ;
```

- Examples:

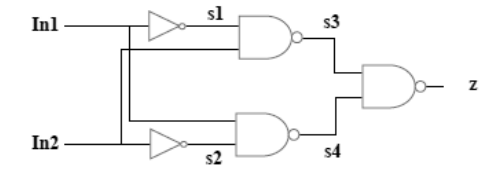
```
zmux : z <= d0 when sell = '0' and sel0 = '0' else
            d1 when sell = '0' and sel0 = '1' else
            d2 when sell = '1' and sel0 = '0' else
            d3 when sell = '1' and sel0 = '1';
```

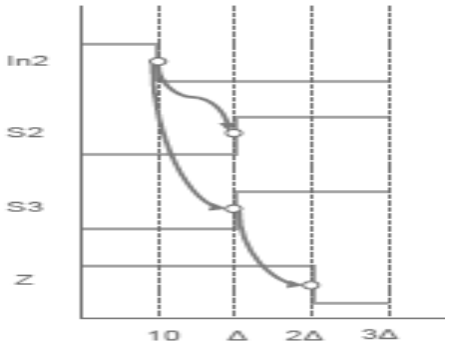
```
gensig: reset <= '1', '0' after 20 ns when aconst else
            '1', '0' after 50 ns;
```

```
Inc: newv <= oldv + 5 after 6 ns;
```

Delta Delays

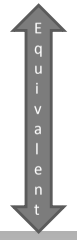
```
architecture behavior of ckt
signal s1, s2, s3, s4:
std_logic:= '0';
begin
    s1 <= not in1;
    s2 <= not in2;
    s3 <= not (s1 and in2);
    s4 <= not (s2 and in1);
    z <= not (s3 and s4);
end architecture behavior;
```





Conditional Signal Assignment Statements

```
zmux: z <= d0 when sell = '0' and sel0 = '0'
else d1 when sell = '0' and sel0 = 'l'
else d2 when sell = '1' and sel0 = '0'
else d3 when sell = '1' and sel0 = '1';
```




```
zmux: process is
begin
        if sell = '0' and sel0 = '0' then z <= d0;
        elsif sell = '0' and sel0 = '1' then z <= dl;
        elsif sell = '1' and sel0 = '0' then z <= d2;
        elsif sell = '1' and sel0 = '1' then z <= d3;
        end if;
        wait on d0, dl, d2, d3, sel0, sell;
end process zmux
```

Conditional Signal Assignment Statements

- Conditional signal assigments may be used to define tri-state buffers, using the std_logic and std_logic_vector type.

```
architecture COND of TRI_STATE is
 signal TRI_BIT: std_logic;
 signal TRI_BUS:
        std_logic_vector(0 to 7);
begin
 TRI_BIT <= BIT_1 when EN_1 = '1'
     else 'Z';
 TRI_BUS <= BUS_1 when EN_2 = '1'
     else (others => 'Z');
end COND;
```

Conditional Signal Assignment Statements (Synthesis)

- Conditional signal assignments are generally synthesisable.
- A conditional signal assignment will usually result in combinational logic being generated. Assignment to 'Z' will normally generate tri-state drivers. Assinment to 'X' may not be supported.
- If a signal is conditionally assigned to itself, latches may be inferred.

Conditional Signal Assignment Statements

- There must be a final unconditional else expression:

```
architecture COND of WRONG is
begin
  Z <= A when X > 5;  --illegal
end COND;
```

Selected Signal Assignment Statements

- A selected signal assignment uses the with
- statement, and must include all possible cases. The others case ensures that all cases are covered.

```
Architecture example of control_stmts is
begin
With sel select
        m <= c when b"00",
        m <= d when b"01",
        m <= a when b"10",
        m <= b when others ;
End example;
```

Conditional Signal Assignment

- Sometimes we may not want to schedule any new transactions on the signal. To do this, use the keyword unaffected instead of a normal waveform.

```
zmux : z <= dO when sel1 = '0' and sel0 = '0' else
        d1 when sel1 = 'O' and sel0 = '1' else
        d2 when sel1 = '1' and sel0 = '0' else
        unaffected when sel1 = '1' and sel0 = '1';
```

```
zmux: process is
begin
        if sell = '0' and sel0 = '0' then z <= d0;
        elsif sell = '0' and sel0 = '1' then z <= d1;
        elsif sell = '1' and sel0 = '0' then z <= d2;
        elsif sell = '1' and sel0 = '1' then null;
        end if;
        wait on d0, d1, d2, d3, sel0, sell;
end process zmux;
```

Selected Signal Assignment Statements

- Syntax:

```
with expression select
name <= [delaymechanism]
                { waveform when choices ,}
                waveform when choices;
```

- Example:

```
alu: with opcode select
                result <= a + b after Tpd when Aadd | Aaddu,
                                a - b after Tpd when Asub | Asubu,
                                a and b after Tpd when Aand,
                                a or b after Tpd when Aor,
                                a after Tpd when ApassA;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity encoder is
port (invec: in std_logic_vector(7 downto 0);
                enc out: out std_logic_vector(2 downto 0)
        );
end encoder;
architecture rtl of encoder is
begin
        enc out <= "111" when invec(7) = '1' else
                        "110" when invec(6) = '1' else
                        "101" when invec(5) = '1' else
                        "100" when invec(4) = '1' else
                        "011" when invec(3) = '1' else
                        "010" when invec(2) = '1' else
                        "001" when invec(1) = '1' else
                        "000" when invec(0) = '1' else
                        "000";
end rtl;
VHD
```

Concurrent Assertion Statements: Example

```
entity SRFilipFlop is
port ( s, r: in bit; q, qb : out bit);
end entity SRFlipFlop;

architecture functional of SRFilipFlop is
begin
    q <= '1' when s = '1', else
         '0' when r = '1';
    qb <= '0' when s = '1' else
          '1" when r = '1';
    check: assert not (s = '1' and r = '1')
    report "Incorrect use of SR flipflop: s and r both
    '1'";
end architecture functional;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY mux2_1_16wide IS
PORT(  in_a : IN STD_LOGIC_VECTOR(15 DOWNTO 0); --input a
       in_b : IN STD_LOGIC_VECTOR(15 DOWNTO 0); --input b
       sel : IN STD_LOGIC; --select input
       output : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) --data output
    );
END mux2_1_16wide;

ARCHITECTURE beh OF mux2_1_16wide IS
BEGIN
    WITH sel SELECT
        output <= in_a WHEN '0',
                  in_b WHEN '1',
                  (OTHERS => 'X') WHEN OTHERS;
END beh;
```

OTHERS is also used to provide a shorthand method of saying, "make all the bits of the target signal 'X" for however many bits are in target signal.
(OTHERS => 'X') WHEN OTHERS

Selected Signal Assignment Statements

```
architecture truthtable of fulladder is
begin
with bit_vector'(a, b, ci) select
   (cout, s) <= bit_vector'("00") when "000",
                Bit_vector'("01") when "001",
                Bit_vector'("01") when "010",
                Bit_vector'("10") when "011",
                Bit_vector'("01") when "100",
                Bit_vector'("10") when "101",
                Bit_vector'("10") when "110",
                Bit_vector'("11") when "111";
end architecture truthtable;
```

Selected Signal Assignment Statements

```
alu: with opcode select
     result <= a + b after Tpd when Aadd | Aaddu,
               a - b after Tpd when Asub | Asubu,
               a and b after Tpd when Aand,
               a or b after Tpd when Aor,
               a after Tpd when ApassA;
```

```
ARCHITECTURE beh OF mux5_1_1wide IS
BEGIN
WITH sel SELECT
z_out <= a_input WHEN "000" | "001" | "111",
b_input WHEN "011" | "101",
c_input WHEN "010",
d_input WHEN "100",
e_input WHEN "110",
'X' WHEN OTHERS;
END beh;
```

```
alu: process is
begin
      case opcode is
            when Aadd | Aaddu => result <= a + b after Tpd;
            when Asub | Asubu => result <= a - b after Tpd;
            when Aand => result <= a and b after Tpd;
            when Aor => result <= a or b after Tpd;
            when ApassA => result <= a after Tpd;
      end case;
      wait on opcode, a, b;
end process alu;
```

Concurrent Assertion Statement: Example

```
entity SRFilipFlop is
port ( s, r: in bit; q, qb : out bit);
begin
      check: assert not (s = ’1’ and r = ’1’)
            report "Incorrect use of SR flipflop: s and r both ’1’"
                  & "at time" & time’image(now) ;
end entity SRFlipFlop;

architecture functional of SRFilipFlop is
begin
      q <= ’1’ when s = ’1’, else
            ’0’ when r = ’1’;
      qb <= ’0’ when s = ’1’ else
            ’1" when r = ’1’;
end architecture functional;
```