VHDL = VHSIC Hardware Description Language

VHSIC = Very High Speed Integrated Circuit

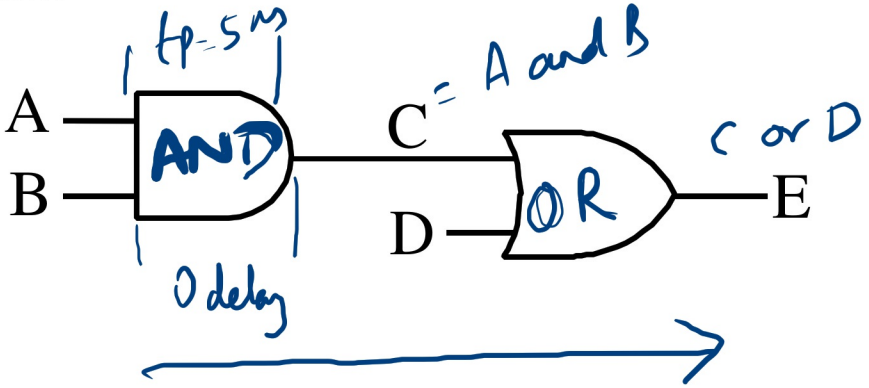
Hardware description, simulation, and synthesis

Describes hardware at different levels:
behavioral, logic equation, structural

Top-down design methodology

Technology Independent

Figure 2-1 Gate Network



Concurrent Statements

```
C <= A and B  after 5 ns;
E <= C or D   after 5 ns;
```

If delay is not specified, "delta" delay is assumed

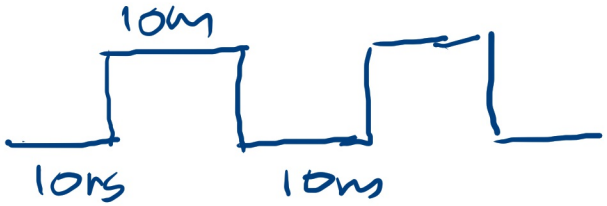
```
C <= A and B;
E <= C or D;
```

Order of concurrent statements is not important

```
E <= C or D;
C <= A and B;
```

This statement executes repeatedly

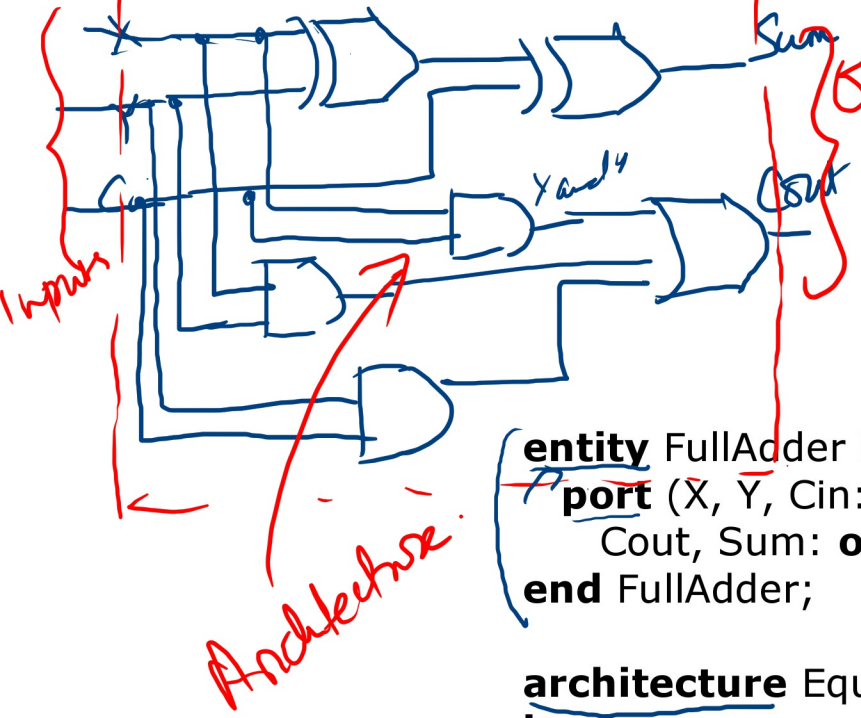
```
CLK <= not CLK after 10 ns;
```

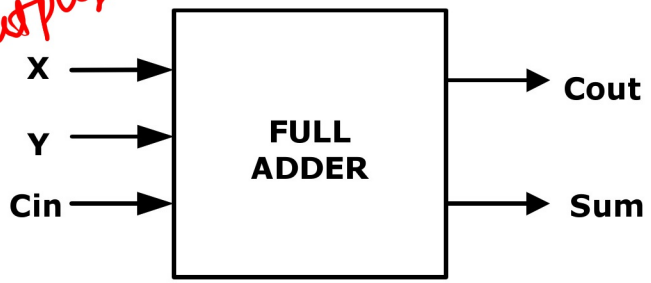


This statement causes a simulation error

```
CLK <= not CLK;
```

From Page 45-46






```
entity FullAdder is
  port (X, Y, Cin: in bit;     -- Inputs
        Cout, Sum: out bit);   -- Outputs
end FullAdder;

architecture Equations of FullAdder is
begin                          -- Concurrent Assignments
  Sum <= X xor Y xor Cin after 10 ns;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations;
```

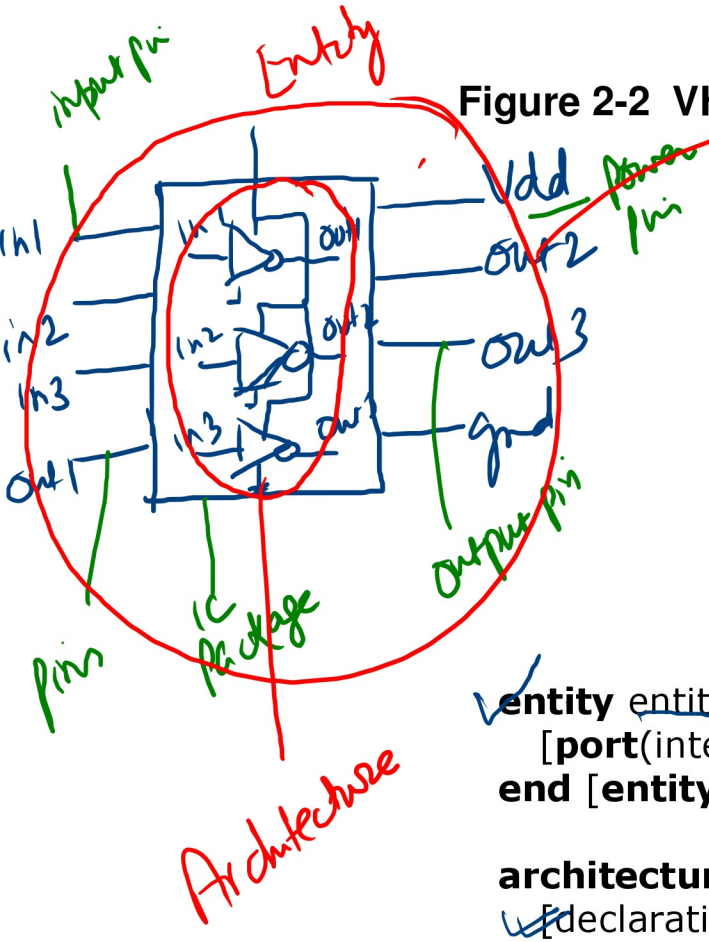
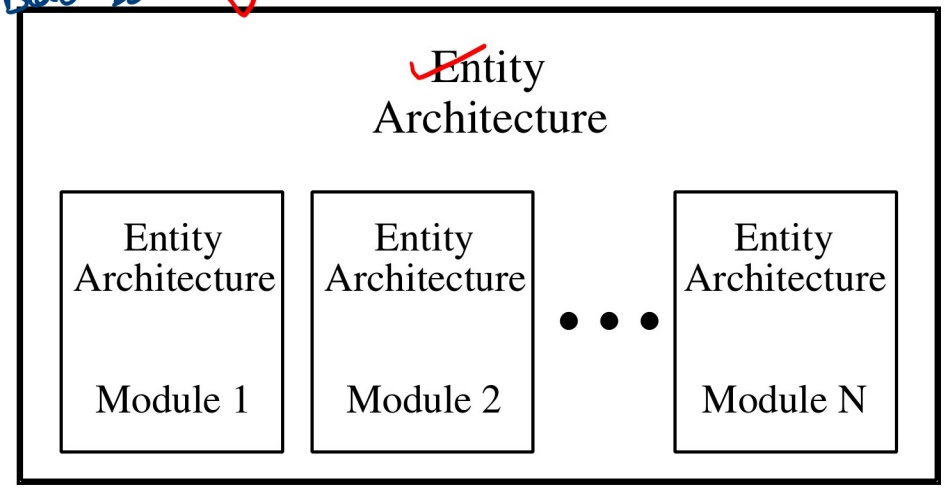


Figure 2-2 VHDL Program Structure



entity entity-name **is**
  [**port**(interface-signal-declaration);]
end [**entity**] [entity-name];

architecture architecture-name **of** entity-name **is**
  [declarations]
begin
  architecture body
end [**architecture**] [architecture-name];

Figure 2-3 4-bit Binary Adder

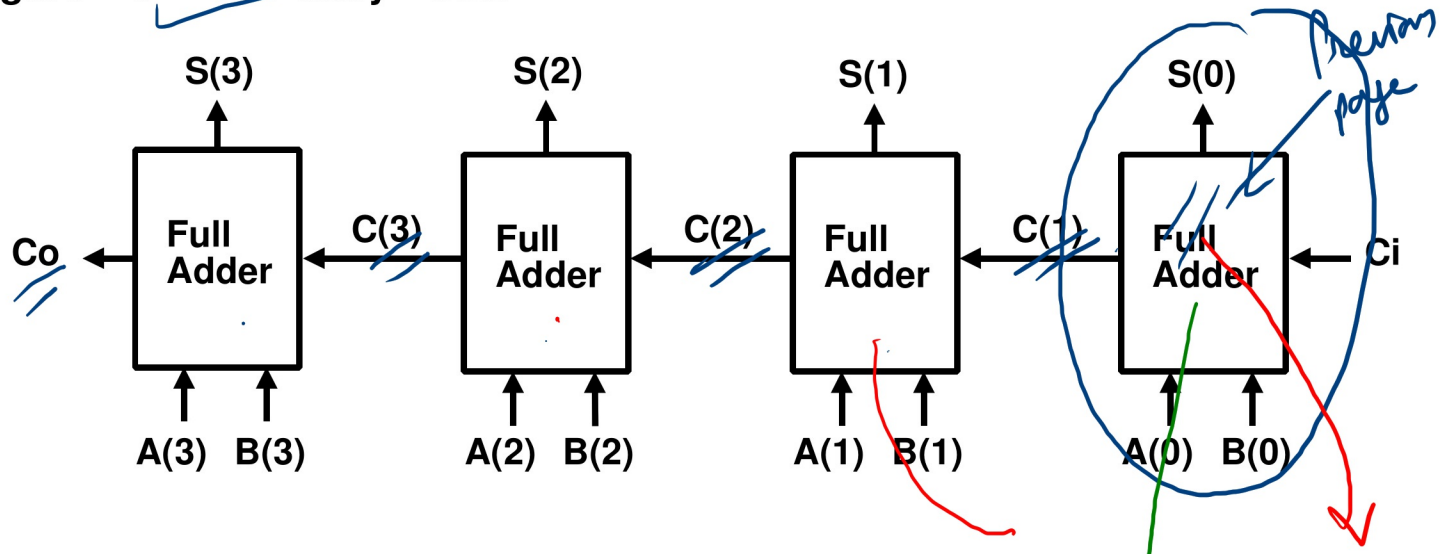


Figure 2-4(i) Structural Description of 4-bit Adder

```
entity Adder4 is
  port (A, B: in bit_vector(3 downto 0); Ci: in bit;      -- Inputs
        S: out bit_vector(3 downto 0); Co: out bit);      -- Outputs
end Adder4;
```

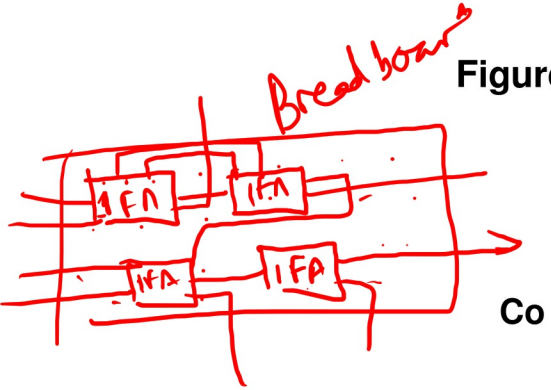


Figure 2-3 4-bit Binary Adder

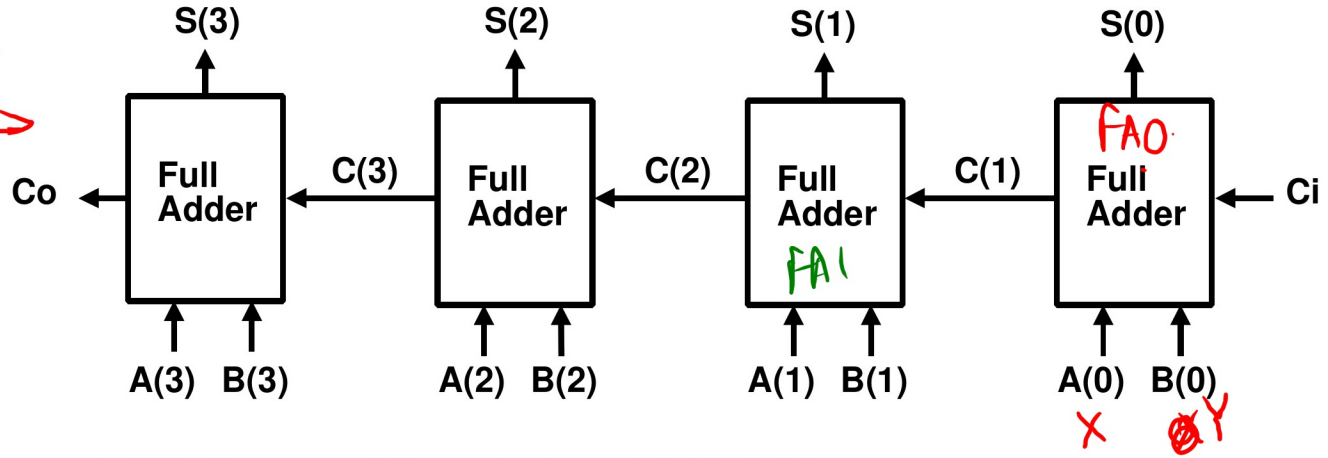


Figure 2-4(ii) Structural Description of 4-bit Adder

```
architecture Structure of Adder4 is
component FullAdder
   port (X, Y, Cin: in bit;          -- Inputs
         Cout, Sum: out bit);        -- Outputs
end component;
signal C: bit_vector(3 downto 1);
begin     --instantiate four copies of the FullAdder
   FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
   FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
   FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
   FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```

One bit full adder is my component.

← Assignment

From Page 49

```
list A B Co C Ci S   -- put these signals on the output list
force A 1111         -- set the A inputs to 1111
force B 0001         -- set the B inputs to 0001
force Ci 1           -- set the Ci to 1
run 50               -- run the simulation for 50 ns
force Ci 0
force A 0101
force B 1110
run 50
```

ns	delta	a	b	co	c	ci	s
0	+0	0000	0000	0	000	0	0000
10	+1	1111	0001	0	000	1	0000
10	+0	1111	0001	0	001	1	1111
20	+0	1111	0001	0	011	1	1101
30	+0	1111	0001	0	111	1	1001
40	+0	1111	0001	1	111	1	0001
50	+0	0101	1110	1	111	0	0001
60	+0	0101	1110	1	110	0	0101
70	+0	0101	1110	1	100	0	0111
80	+0	0101	1110	1	100	0	0011

VHDL Processes

General form of Process

```
process(sensitivity-list)
   begin
      sequential-statements
   end process;
```

Process example

```
process (B, C, D)
begin
   A <= B;   -- statement 1
   B <= C;   -- statement 2
   C <= D;   -- statement 3
end process;
```

Simulation results

time	delta	A	B	C	D	
0	+0	1	2	3	0	
10	+0	1	2	3	4	(statements 1,2,3 execute; then update A,B,C)
10	+1	2	3	4	4	(statements 1,2,3 execute; then update A,B,C)
10	+2	3	4	4	4	(statements 1,2,3 execute; then update A,B,C)
10	+3	4	4	4	4	(no further execution occurs)

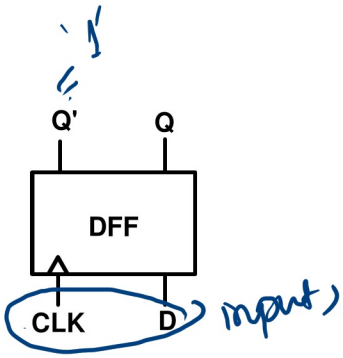
Concurrent Statements

```
A <= B;   -- statement 1
B <= C;   -- statement 2
C <= D;   -- statement 3
```

Simulation Results

time	Δ	A	B	C	D	
0	+0	1	2	3	0	
10	+0	1	2	3	4	(statement 3 executes first)
10	+1	1	2	4	4	(then statement 2 executes)
10	+2	1	4	4	4	(then statement 1 executes)
10	+3	4	4	4	4	(no further execution occurs)

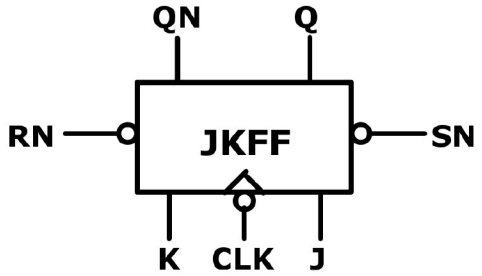
Figure 2-5 D Flip-flop Model




```
entity DFF is
  port (D, CLK: in bit;
        Q: out bit;  QN: out bit := '1');
-- initialize QN to '1' since bit signals are initialized to '0' by default
end DFF;

architecture SIMPLE of DFF is
begin
    process (CLK)                    -- process is executed when CLK changes
    begin
       if CLK = '1' then             -- rising edge of clock
         Q  <= D after 10 ns;
         QN  <= not D after 10 ns;
       end if;
    end process;
end SIMPLE;
```

Figure 2-7 J-K Flip-flop Model

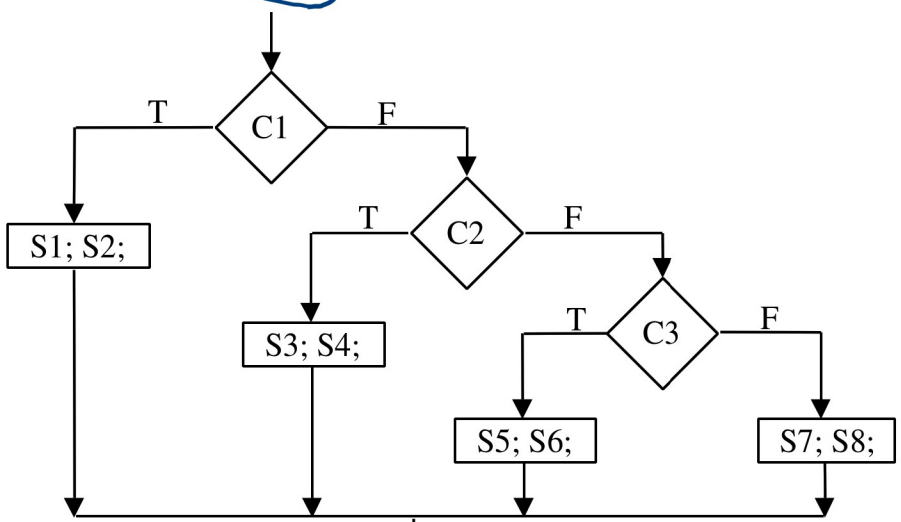



```
entity JKFF is
   port (SN, RN, J, K, CLK: in bit;                -- inputs
         Q: inout bit; QN: out bit := '1');        -- see Note 1
end JKFF;

architecture JKFF1 of JKFF is
begin
   process (SN, RN, CLK)                           -- see Note 2
   begin
      if RN = '0' then  Q<= '0' after 10 ns;       -- RN=0 will clear the FF
      elsif SN = '0'  then  Q<= '1' after 10 ns;   -- SN=0 will set the FF
      elsif CLK = '0' and CLK'event then           -- see Note 3
         Q <= (J and not Q) or (not K and Q) after 10 ns;   -- see Note 4
      end if;
   end process;
   QN <= not Q;                                    -- see Note 5
end JKFF1;
```

Note 1: Q is declared as inout (rather than out) because it appears on both the left and right sides of an assignment within the architecture.
Note 2: The flip-flop can change state in response to changes in SN, RN, and CLK, so these 3 signals are in the sensitivity list.
Note 3: The condition (CLK = '0' and CLK'event) is TRUE only if CLK has just changed from '1' to '0'.
Note 4: Characteristic equation which describes behavior of J-K flip-flop.
Note 5: Every time Q changes, QN will be updated. If this statement were placed within the process, the old value of Q would be used instead of the new value.

Figure 2-8 Equivalent Representations of a Flowchart Using Nested Ifs and Elsifs




```
if (C1) then S1; S2;
  else if (C2) then S3; S4;
         else if (C3) then S5; S6;
                else S7; S8;
              end if;
       end if;
end if;
```

```
if (C1) then S1; S2;
  elsif (C2) then S3; S4;
  elsif (C3) then S5; S6;
  else S7; S8;
end if;
```

(2)

In VHDL there are 3 types of delay:

- Inertial Delay — default
- Transport delay
- (Delta Delay)

* Inertial delays are the default delays in VHDL.
  Intended to model gates and devices that do not propagate short pulses from the i/p to the o/p.
* Transport delay — intended to model the delay introduced by wiring.
  This you have to specify

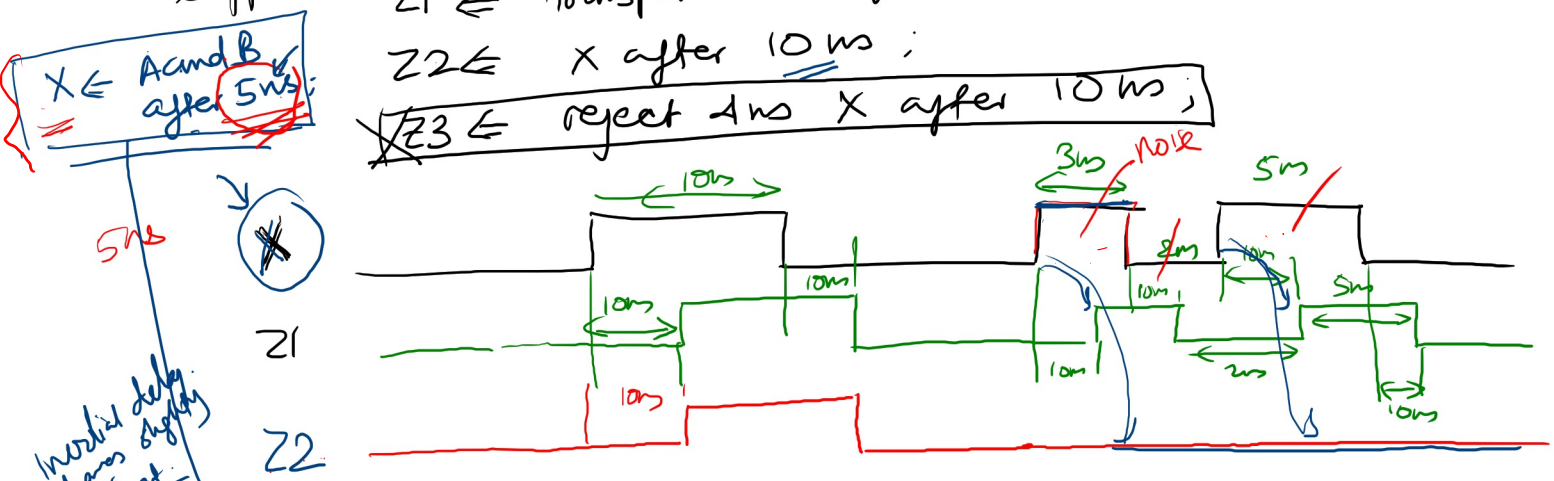
↳ Delta delay → Delay at which the simulation time advances. (more on this later).

X is an input

Suppose

```
Z1 <= transport X after 10 ns;
Z2 <= X after 10 ns;
Z3 <= reject 4ns X after 10 ns;
```

```
X <= A and B after 5ns;
```



Inertial delay behaves slightly different.