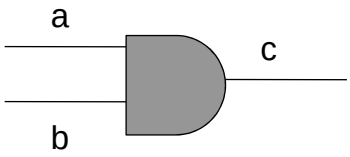
VHDL

Behavioral Modeling

- Signal assignment a<= b, executed if b changes
- Signal b is in sensitivity list of statement
- Signal assignment causes a *transaction*
- If different value of 'a', *transaction* causes event to be scheduled for target signal 'a'
- No change in value of ‘a’ – no event but transaction
- C <= a and b after 10ns- result scheduled after 10 ns from computation of new value
- Change in value ->Transaction schedules an event

Transaction and Event

- And gate
- Value of c may change depending on change in a or b
- If a = ‘0’, b changes from ‘1’ to ‘0’
- No change in 0
- Transaction occurs on *c*. It causes no event to be scheduled on *c*



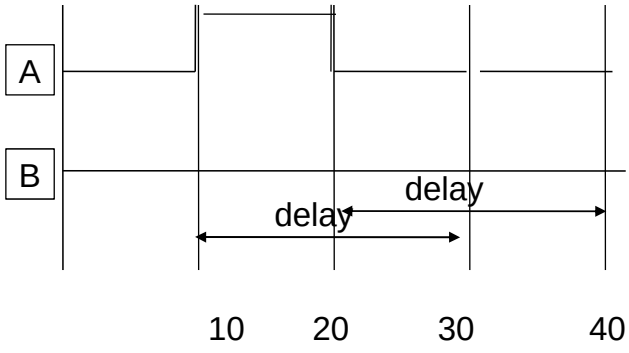
Concurrent statement

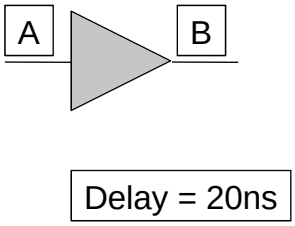
```
ARCHITECTURE arch OF mux IS
  SIGNAL sel : INTEGER;
BEGIN
 WITH sel SELECT
x <= a AFTER  0.5 NS WHEN 0 ,
        b AFTER  0.5 NS WHEN 1,
        c AFTER  0.5 NS WHEN 2 ,
        d AFTER  0.5 NS WHEN 3 ;

  sel <= 0 WHEN s0 = '0' AND s1 = '0' ELSE
              1 WHEN s0 = '1' AND s1 = '0' ELSE
              2 WHEN s0 = '0' AND s1 = '1' ELSE
              3;
```

Transport Vs. Inertial Delay

- Inertial delay (default) - behaves as in device
- o/p signal has inertia- need to overcome
- Inertial delay = delay through device
- Signal value maintained for longer than the device delay
- Spikes have periods less than delay, output should not change
- Simulator swallows the change

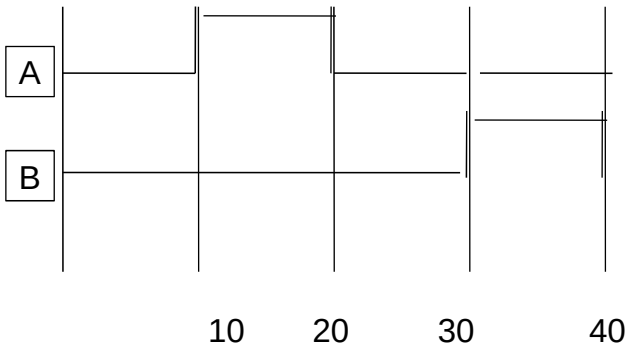


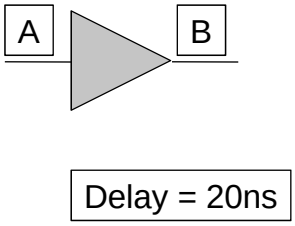


Transport Vs. Inertial Delay

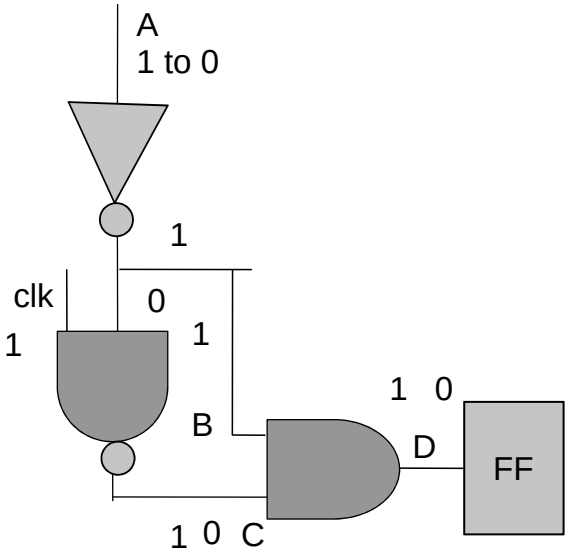
Transport delay

- Modeling wire delays
- Events are ordered before propagation
- B <= *transport* A after 20 ns





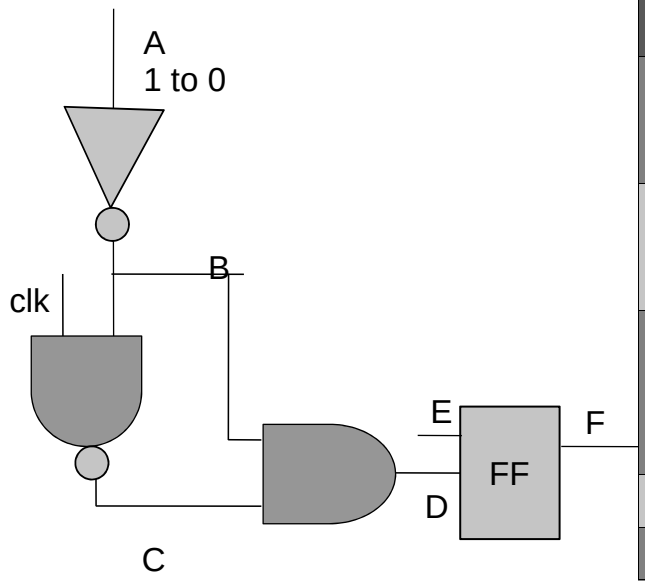
Simulation deltas (non delta delay)



AND first	**NAND first**
Evaluate NOT	Evaluate NOT
B <= 1	B <=1
Evaluate AND (C=1)	Evaluate NAND
D <=1	C <=0
Evaluate NAND	Evaluate AND
C<=0	D <=0
Evaluate AND D <=0	

- Used to order events during simulation for consistency
- Zero delay events
- Simulation delta delay prevents this

Simulation deltas (delta delay)



Time	Delta time	Activity
10ns	1	A<=0 Evaluate NOT
	2	B <=1 Evaluate AND NAND
	3	D <=1 C<=0 Evaluate AND
	4	D <=0
11ns		

- Independent of evaluation order
- Delta delay is used when 0 delay is specified
- Delta delay is very small amount of time
- A <= B after 0 ns;

Drivers

- Data bus is modeled by multiply driven signals
- Multiply driven signal has many drivers
- A driver contributes to the value of a signal
- Value of all drivers is resolved by resolution function
- A <= b after 10ns;
- A <= c after 10ns;
- A is driven by two sources
- Q <= a when s0 = '0' and s1 = '1' else '0';
- Q <= b when s0 = '0' and s1 = '1' else '0';
- Q <= c when s0 = '0' and s1 = '1' else '0';
- Q <= d when s0 = '0' and s1 = '1' else '0';

Generics

- Pass information to an instance of an entity
- Delay times, datapath width
- Model elaboration – linked to simulator

```
ENTITY and2 IS
GENERIC(del: TIME );
PORT(a,b: IN STD_LOGIC; c : OUT STD_LOGIC);
END and2
ARCHITECTURE and2 OF and2 IS
 C <= a AND  b AFTER del;
END and2
```

Generics (Contd)

```
ENTITY test IS
PORT (ina,inb : IN STD_LOGIC; x: OUT STD_LOGIC);
END test;
ARCHITECTURE test OF test IS
COMPONENT and2
GENERIC(del: TIME := 15ns);
PORT(a,b: IN STD_LOGIC; c : OUT STD_LOGIC);
END COMPONENT;
SIGNAL i1: STD_LOGIC;
BEGIN
A1: and2 GENERIC MAP(10 ns)
PORT MAP(ina,inb,i1);
A2: and2
PORT MAP(i1,inb,x);
END test
```

Block statement

- Logically group areas of model
- Same scope rules as in C
- Global signal access by scope resolution
- PORT MAP allowed

```
ARCHITECTURE exblock OF ex IS
SIGNAL a,b : BIT
BEGIN
ALU : BLOCK
SIGNAL a;
BEGIN
---
END BLOCK ALU;
reg : BLOCK
SIGNAL a;
BEGIN
---
END BLOCK reg;
END
```

Sequential Processing

- PROCESS statement
- Sensitivity list or WAIT stmt
- Signal and variable
- Variables are for local storage
- Signals are interconnections
- Variable assignment is done immediately
- Variables are used for pre computations

Sequential Statements

- IF .. THEN .. ELSIF … ELSE .. END IF;
- CASE .. IS.. WHEN ..=> ...; WHEN OTHERS... END CASE
- WHILE .... LOOP … END LOOP;
- FOR .. IN .. TO .. LOOP … END LOOP;
- NEXT .. (continue in C)
- EXIT (break in C)
- ASSERT – report textual info
- WAIT – allows suspension of sequential stmt

ASSERT

- Checks value of boolean exp.
- If false , o/p to terminal
- Severity level – note, warning, error, failure
- Note – progress of loop
- Warning - Signal appeared not matching with expected
- Error - +ve value instead of negative
- Fail – divide by 0
- No h/w built

```
IF (clk = '1') THEN
 ASSERT ( last_val >= 20 )
   REPORT “wrong”
   SEVERITY WARNING'
END IF
```

WAIT STMT

- Suspends sequential execution
- Resume execution by
  - WAIT ON signal
  - WAIT UNITIL exp
  - WAIT FOR time
- Exp requires atleast one signal
- Specifying clk i/p to synthesis tool
- Delay process execution
- Modify sensitivity list dynamically

```
PROCESS
BEGIN
 WAIT UNTIL clk = '1' AND
   clk'EVENT;
Q <= D;
END PROCESS;
```

Attribute of a signal

- Generates FF
- Q <=D at rising edge of clk
- Synthesis : D FF with no set/reset

WAIT STMT (contd)

- Synchronous reset

```
PROCESS
BEGIN
 WAIT UNTIL clk = '1' AND
    clk'EVENT;
IF (reset = '1' ) THEN Q <= '0'
ELSE Q <= D;
END IF
END PROCESS;
```

- Asynchronous reset

```
PROCESS
BEGIN
IF (reset = '1' ) THEN Q <= '0'
 ELSIF clk'EVENT AND clk = '1'
    THEN Q <= D;
END IF
WAIT ON reset, clock
END PROCESS;
```

WAIT (contd.)

- Multiple WAIT

WAIT ON a UNTIL (var = 5) FOR 5 usec ;

- WAIT time-out
  - To avoid wait for ever
  - Two communicating process
  - Process A waits for signal from process B and vice-versa
  - Use WAIT UNTIL sendb = '0' FOR 20 ns;
- Either sensitivity list or WAIT
- Simulator executes the process once during initialization
  - WAIT stmt at the end

Signal Assignment in PROCESS

- Concurrent assignment problem
- Value assigned to last signal assignment does not appear
- Signal assignment schedules event at next delta time point
- Next delta time is when the process finishes
- Thus wrong value of signal
- Use variables, immediate assignment

```
SIGNAL sel : INTEGER
PROCESS(a,b,c,d, s0,s1)
BEGIN
  Sel <= 0;
IF (s0 = '1' THEN sel <= sel + 1; END IF;
IF (s1 = '1' THEN sel <= sel + 2; END IF;

CASE sel IS
WHEN 0 => x <= a;
.......
END CASE
END PROCESS
```

Solutions: Signal Assignment in PROCESS

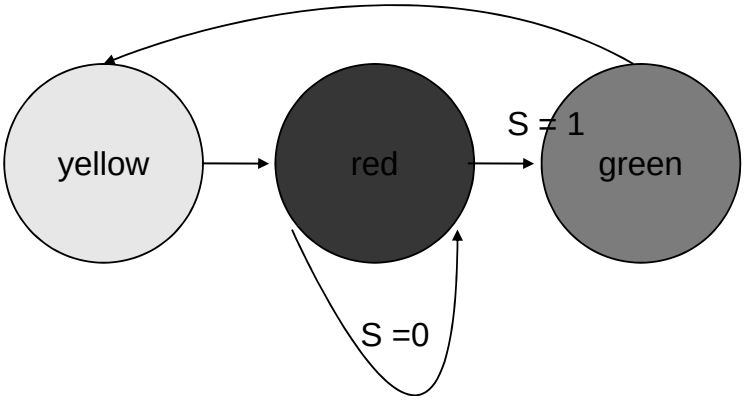
Solution: 1

```
SIGNAL sel : INTEGER
PROCESS
BEGIN
  Sel <= 0;
WAIT FOR 0 ns;
IF (s0 = '1' THEN sel <= sel + 1; END IF;
WAIT FOR 0 ns;
IF (s1 = '1' THEN sel <= sel + 2; END IF;
WAIT FOR 0 ns;
CASE sel IS
WHEN 0 => x <= a;
.......
END CASE;
WAIT on a,b,c,d, s0,s1;
END PROCESS
```

Solution: 2

```
VARIABLE sel : INTEGER
PROCESS(a,b,c,d,s0,s1)
BEGIN
  Sel := 0;
IF (s0 = '1' THEN sel <= sel + 1; END IF;
IF (s1 = '1' THEN sel <= sel + 2; END IF;
CASE sel IS
WHEN 0 => x <= a;
.......
END CASE;
END PROCESS
```

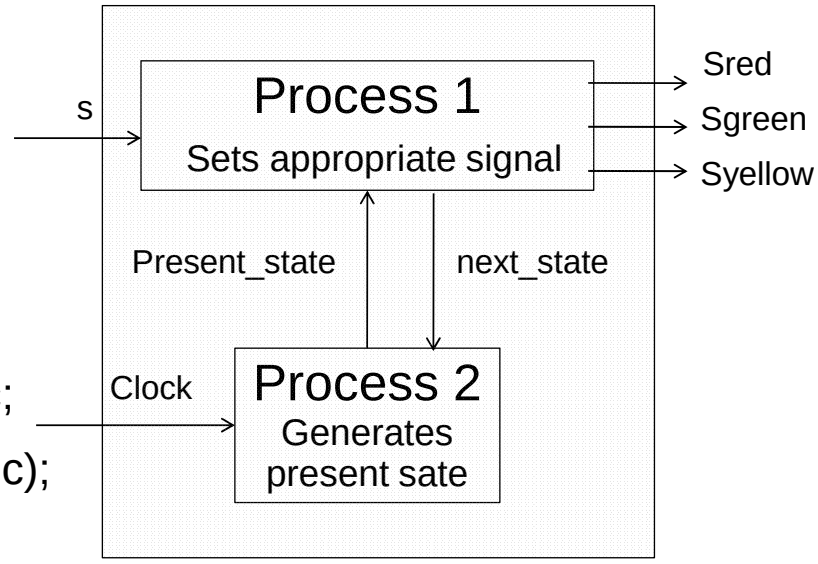
Example: State Machine



- Three output signals: red, yellow, green to be set depending on the present signal state and a sensor
- With tick of a clock the signal goes to the next state and sets output signal accordingly

Entity: trafsig

```
ENTITY trafsig IS
PORT (s: IN std_logic;
      Clock: IN std_logic;
      Sred: OUT std_logic;
      sgreen: OUT std_logic;
      syellow : OUT std_logic);
END trafsig
```



Architecture

Architecture: Process 1

```
Architecture traf of trafsig is
TYPE  t_state is (red,green,yellow);
Signal present, next: t_state;
BEGIN
PROCESS(present_state,s)
BEGIN
CASE present IS
        WHEN green =>
                Next <= yellow;
                sred = '0';
                syellow = '0';
                sgreen = '1'
```

```
WHEN red =>
        IF (s ='1') THEN next <= green;
        else next <= red;
        sred = '1';
        syellow = '0';
        sgreen = '0' ;
WHEN yellow =>
        next <= red;
        sred = '0';
        syellow = '1';
        sgreen = '0';
End CASE
End PROCESS
```

Architecture: Process 2

```
PROCESS
BEGIN
 WAIT UNTIL clock'event and clock = '1';
 Present <= next;
End process;
End trafsig
```

Or

```
PROCESS (clock)
BEGIN
 if  clock'event and clock = ‘1 then
          present <= next;
End PROCESS;
End trafsig
```

VHDL Object

- VHDL Object types
  - Signal – interconnection
  - Variable – storage
  - Signal, variable can be initialized
  - Constants – values
- CONSTANT PI : REAL := 3.1424
- Variables in process, subprogram
- Signal entity, architecture, package

DATA TYPES

- Scalar – integer, real, enumerated, physical
- Composite – array, record
- Access -
- File -

DATA TYPES

Enumerated data type

- TYPE inst IS (add, sub,lda);

CASE inst

WHEN add => a:= data;

WHEN sub => b := data;

END CASE

- TYPE val IS ('0', '1', 'X', 'Z');

Physical data type

TYPE current IS RANGE 0 TO 1000000

UNITS

na;

ua = 10000 na;

ma = 1000 ua;

a = 1000 ma;

END UNITS;

out_current <= 100 ma;

DATA TYPES

- Composite: array, record
- Arrays : RAM, ROM modeling
- Record : data packet, instruction

TYPE data_bus IS ARRAY (0 TO 31) OF BIT

VARIABLE X: data_bus;

VARIABLE Y : BIT;

Y = X(0);