# Sequential Statements

- **Variable assignment statement**
- **Signal assignment**
- **If statement**
- **Case statement**
- **Loop statement**
- **Next statement**

- Exit statement
- Null statement
- Procedure call statement
- Return statement
- Assertion statement

- Variable assignment statement
- Signal assignment

# Variable assignment statement

**Variable_assignment_statement ::= target:=expression;**

```
architecture RTL of VASSIGN is
    signal A, B, J  : bit_vector(1 downto 0);
    signal E, F, G  : bit;
    begin
        p0 : process (A, B, E, F, G, J)
            variable C, D, H, Y : bit_vector(1 downto 0);
            variable W, Q       : bit_vector(3 downto 0);
            variable Z          : bit_vector(0 to 7);
            variable X          : bit;
            variable DATA       : bit_vector(31 downto 0);
         begin  ...
         end process
     end RTL;
```

# Variable assignment statement

```
p0 : process (A, B, E, F, G, J)
-- A, B, J, D, H  : bit_vector          -- E, F, G  : bit
  begin
    C          := "01";
    X          := E nand F;
    Y          := H or J;
    Z(0 to 3)  := C & D;
    Z(4 to 7)  := (not A) & (A nor B);
    D          := ('1', '0');
    W          := (2 downto 1 => G, 3 => '1', others => '0');
    DATA       := (others => '0');
  end process;
```

The same signal G goes to two bits

Make note of mapping notation again

# Signal assignment statement

VHDL syntax description in metalanguage

**Signal_assignment_statement** ::=

target<=[**transport**]waveform_element{,waveform_element};

waveform_element::=

value_expression[**after** time_expression]|**null**[after time_expression]

# Signal assignment statement

```
p0 : process (A, B)
  begin
    Y <= A nand B after 10 ns;
    X <= transport A nand B
    after 10 ns;
  end process;
p1 : process
  begin
    A <= '0', '1' after 20 ns, '0'
    after 40 ns,  '1' after 60 ns;
    B <= '0', '1' after 30 ns, '0'
    after 35 ns, '1' after 50 ns;
    wait for 80 ns;
  end process;
```

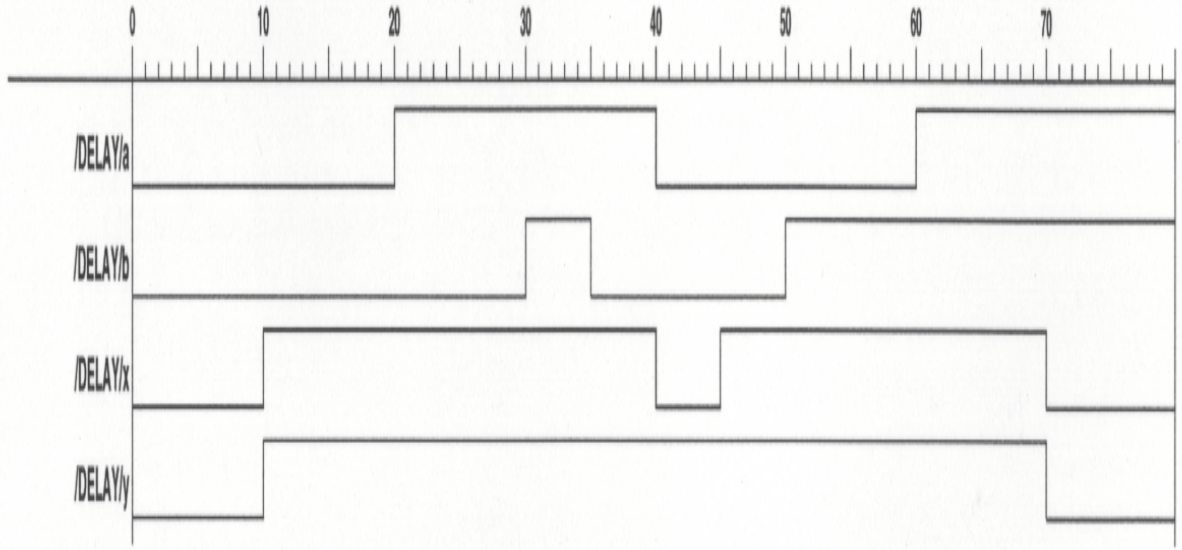



FIGURE 4.1 Inertial and transport delay.

# Signal assignment statement

- The optional keyword transport specifies a *transport delay* rather than an *inertial delay*.
  - **Inertial delays** are characteristic of switching circuits.
  - A pulse with a duration shorter than the switching time of the circuit will not be transmitted in **transport.**

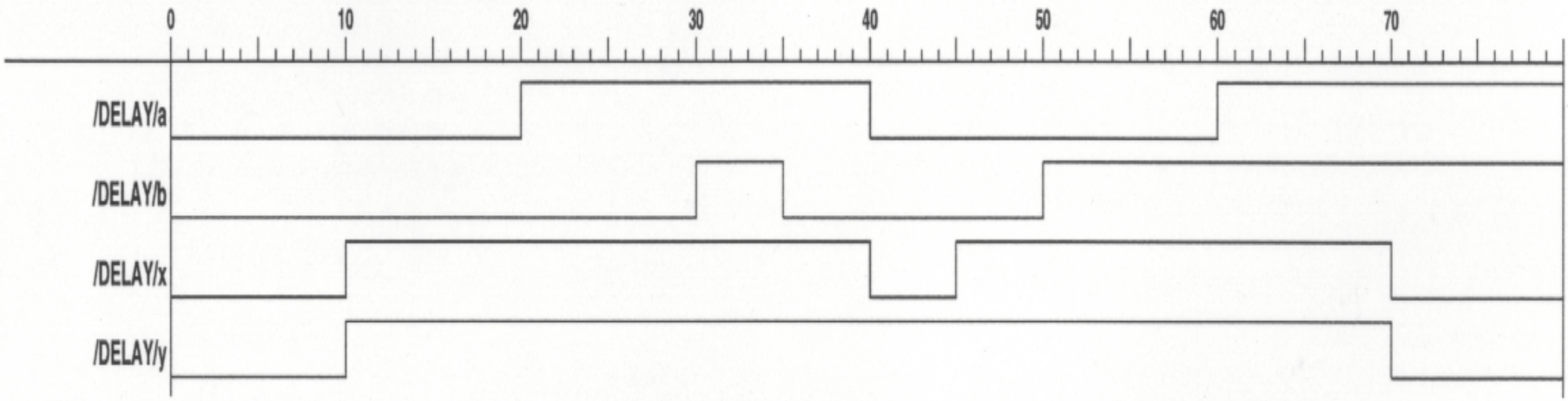



FIGURE 4.1 Inertial and transport delay.

```
entity DELAY is
end DELAY;
architecture RTL of DELAY is
  signal A, B, X, Y : bit;
begin
  p0 : process (A, B)
  begin
    Y <= A nand B after 10 ns;
    X <= transport A nand B after
    10 ns;
  end process;
```

```
p1 : process
  begin
    A <= '0', '1' after 20 ns,
         '0' after 40 ns, '1' after 60 ns;
    B <= '0', '1' after 30 ns,
         '0' after 35 ns, '1' after 50 ns;
    wait for 80 ns;
  end process;
end RTL;
```

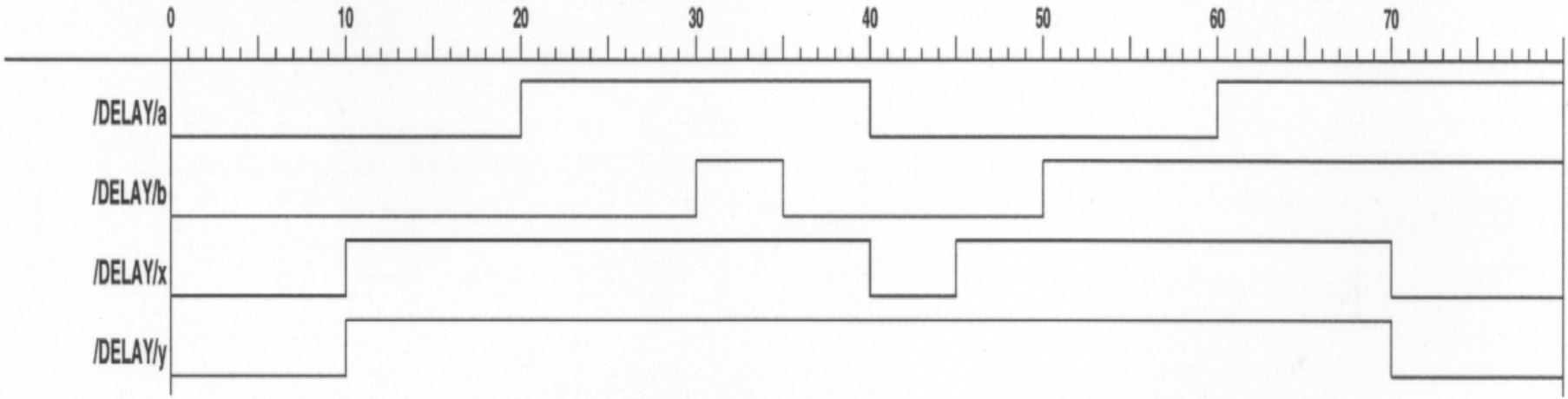



FIGURE 4.1 Inertial and transport delay.

# Signal assignment statement

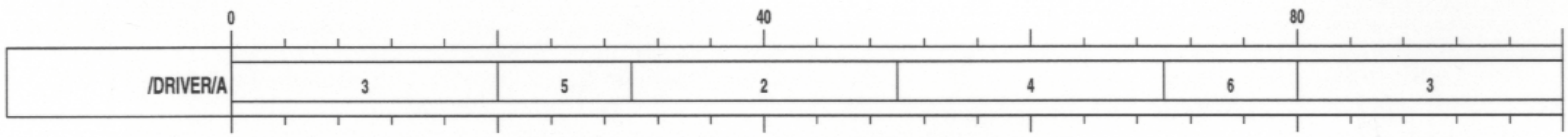



FIGURE 4.2 Simulation waveform for a signal driver.

```
entity DRIVER is
end DRIVER;
architecture RTL of DRIVER is
   signal A : integer;
begin
   pa : process
begin
```

```
A <= 3, 5 after 20 ns, 7 after 40
     ns, 9 after 60 ns;
     wait for 30 ns;
     A <= 2, 4 after 20 ns, 6 after
     40 ns, 8 after 60 ns;
     wait for 50 ns;
   end process;
end RTL;
```

**discarded**

**discarded**

# Signal assignment statement

- Differences between **variables** and **signals**

- 1. Where declared
  - **Local variables** are declared and only visible inside a process or a subprogram.
  - **Signals** cannot be declared inside a process or a subprogram.

- 2. When updated
  - A local variable is immediately updated when the variable assignment statement is executed.
  - A signal assignment statement updates the signal driver.
  - The new value of the signal is updated when the process is suspended.

# Signal assignment statement

3. Variables are cheaper to implement in VHDL simulation since the evaluation of drivers is not needed. They require less memory.

4. Signals communicate among concurrent statements. Ports declared in the entity are signals. Subprogram arguments can be signals or variables.

5. A signal is used to indicate an interconnect (net in a schematic). A local variable is used as a temporary value in a function description.

# Signal assignment statement

6. A local variable is very useful to factor out common parts of complex equations to reduce the mathematical calculation.

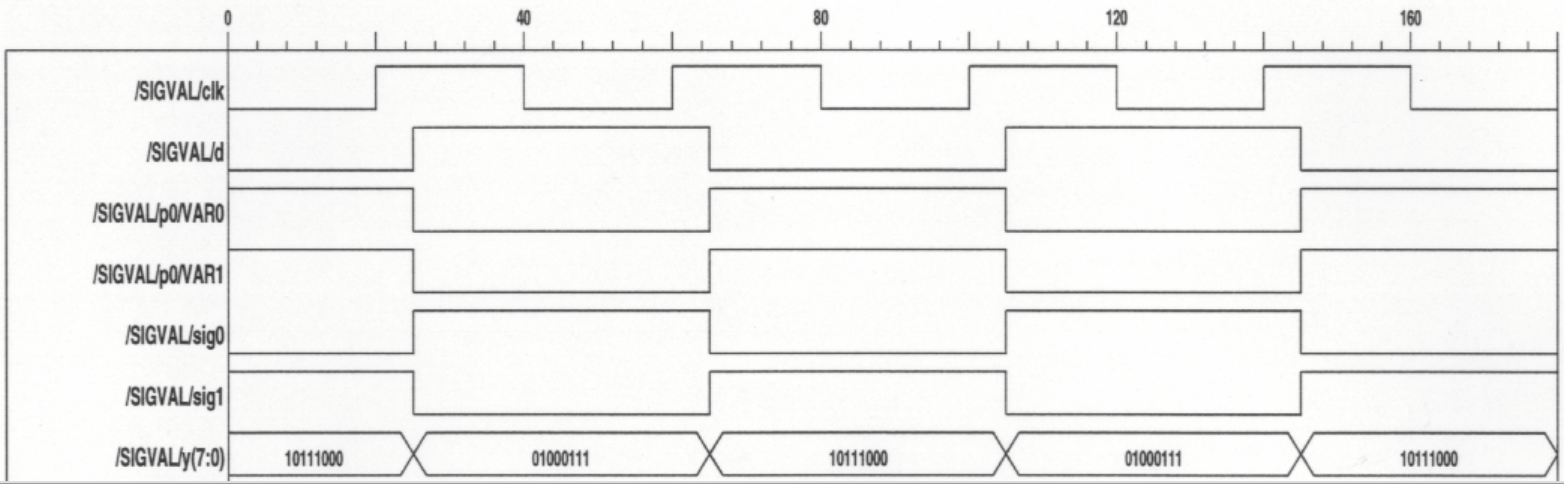
7.

- The right-hand side of a variable assignment statement is an expression.
- There is no associated time expression.
- The right-hand side of a signal assignment statement is a *sequence of waveform elements with associated time expressions.*

# Signal assignment statement

```
entity SIGVAL is
  port (
    CLK, D   : in  bit;
    FF2, FF3 : out bit;
    Y   : out bit_vector(7 downto 0));
end SIGVAL;
architecture RTL of SIGVAL is
  signal FF1, SIG0, SIG1    : bit;
begin
  p0 : process (D, SIG1, SIG0)
    variable VAR0, VAR1 : bit;
begin
    VAR0 := D;
    VAR1 := D;
    SIG0 <= VAR0;
    SIG1 <= VAR1;
    Y(1 downto 0) <= VAR1 & VAR0;
    Y(3 downto 2) <= SIG1 & SIG0;
    VAR0 := not VAR0;
    VAR1 := not VAR1;
    SIG0 <= not VAR0;
    SIG1 <= not D;
    Y(5 downto 4) <= VAR1 & VAR0;
    Y(7 downto 6) <= SIG1 & SIG0;
end process;
```

# Signal assignment statement




C
D
V0
V1
S0
S1
Y

Y <= (S1, S0, ~D, ~D, S1, S0, D, D)

# Signal assignment statement

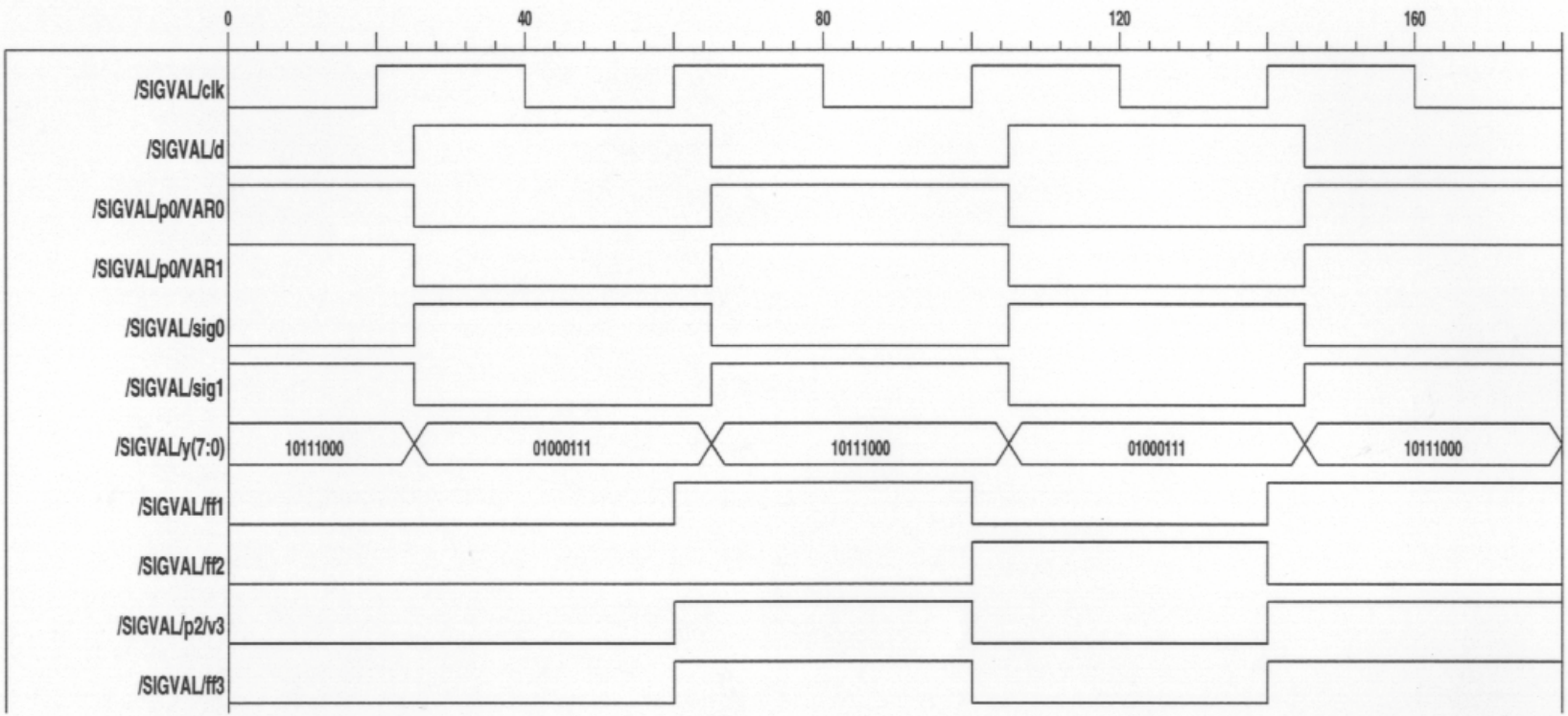



FIGURE 4.3 Simulation waveform for variables and signals.

C
D
V0
V1
S0
S1
Y
F1
F2
V3
F3

# Signal assignment statement

```
p1 : process
  begin
    wait until CLK'event and CLK = '1';
    FF1 <= D;  FF2 <= FF1;
  end process;
  p2 : process
    variable V3 : bit;
  begin
    wait until CLK'event and CLK = '1';
    V3  := D;  FF3 <= V3;
  end process;
end RTL;
```

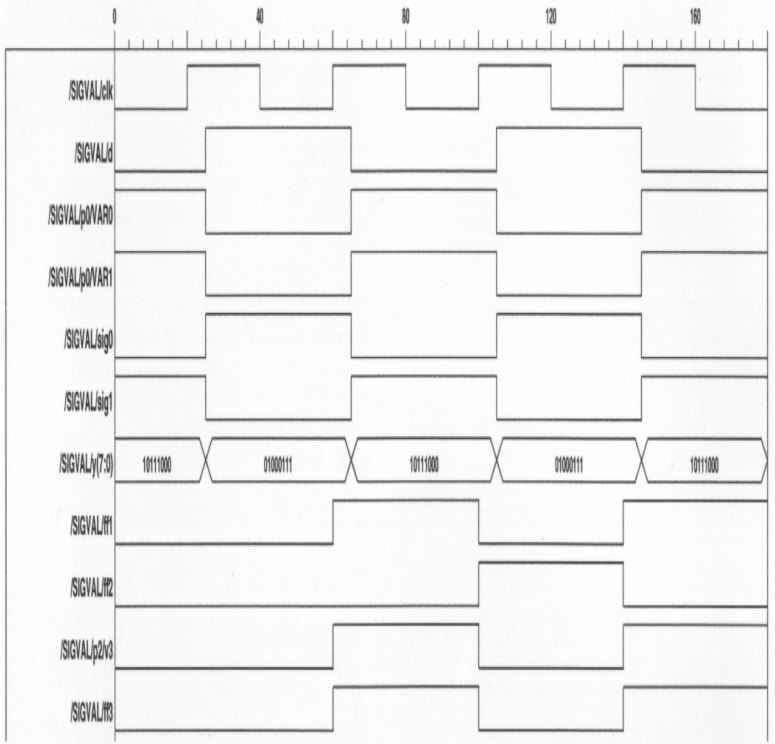



FIGURE 4.3 Simulation waveform for variables and signals.

C
D
V0
V1
S0
S1
Y
F1
F2
V3
F3

# •Signal assignment statement

```
entity TEMP is
end TEMP;
architecture RTL of TEMP is
  signal A, B, C, D, E, F, G, Y, Z : integer;
begin
  p0 : process (A, B, C, D, E, F, G)
  begin
    Y <= A + (B*C + D*E*F + G);
    Z <= A - (B*C + D*E*F + G);
  end process;
end RTL;
architecture RTL1 of TEMP is
signal A, B, C, D, E, F, G, Y, Z : integer;
begin
  p0 : process (A, B, C, D, E, F, G)
variable V : integer;
begin
   V := (B*C + D*E*F + G);
   Y <= A + V; Z <= A - V;
end process;
end RTL1;
architecture RTL2 of TEMP is
   signal A, B, C, D, E, F, G, Y, Z : integer;
   signal V : integer;
begin
   p0 : process (A, B, C, D, E, F, G)
   begin
      V <= (B*C + D*E*F + G);
      Y <= A + V; Z <= A - V;
   end process;
end RTL2;
```