Variable assignment
statement
Signal assignment
wait
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

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 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;

signal A, B, J : bit_vector(1 downto 0);
signal E, F, G : bit;

Variable assigned to a signal

The same signal G (a bit) goes to two bits

Make note of mapping notation again
Formal Syntax of a signal assignment statement

\[
\text{Signal\_assignment\_statement ::= target} \leq [\text{transport}] \text{waveform\_element}\{,\text{waveform\_element}\};
\]

\[
\text{waveform\_element ::= value\_expression[after time\_expression]|[null][after time\_expression]}
\]
Signal assignment statements and wait for

\[ p0 : \text{process} (A, B) \]
\[ \begin{align*}
  Y & \leq A \text{ nand } B \text{ after } 10 \text{ ns;} \\
  X & \leq \text{transport } A \text{ nand } B \text{ after } 10 \text{ ns;}
\end{align*} \]
end process;

\[ p1 : \text{process} \]
\[ \begin{align*}
  A & \leq '0', '1' \text{ after } 20 \text{ ns, } '0' \text{ after } 40 \text{ ns, } '1' \text{ after } 60 \text{ ns;} \\
  B & \leq '0', '1' \text{ after } 30 \text{ ns, } '0' \text{ after } 35 \text{ ns, } '1' \text{ after } 50 \text{ ns;} \\
  \text{wait for } 80 \text{ ns;}
\end{align*} \]
end process;

Recall waveforms, transport and inertial delay

A pulse with a duration shorter than the switching time of the circuit will not be transmitted in transport.
Signal assignment statement

Inertial and Transport Delays
entity DELAY is  
end DELAY;  
arbitrary 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;  
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.
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;

This slide explains the role of **wait for** to discard part of 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.
Differences between **variables** and **signals**

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

7. Right-hand sides:
   ➢ 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.
entity SIGVAL is
            port (               begin
                CLK, D : in bit;       VAR0 := D;
                FF2, FF3 : out bit;     VAR1 := D;
                Y : out bit_vector(7 downto 0));   SIG0 <= VAR0;
            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;
        Variables on left
        Variables on right
Simulation waveform for variables and signals

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;

Y <= (S1, S0, ~D, ~D, S1, S0, D, D)
**Timing of variables versus timing of signals**

```vhdl
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;
```

- **FF2 is old value of FF1 according to signal semantics**
- **Variable V3 changes at the same time as FF1, and so FF3**
- **FF3 unlike FF2**

**MORAL:** Signals are scheduled, variables change immediately.
Three architectures

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;
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 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;

First architecture has no variables
Second architecture uses variable V
Third architecture uses additional signal V

Their operation is different because signal V is scheduled and variable immediately assigned

v calculated immediately

The same statements

Uses old value of v, because it is a signal
Sources

➤ VLSI. Ohio University, Starzyk