VHDL Literals

- ## Numbers

  ```
  14364
  14_364
  16#FF03#
  16.3
  13_634.34
  1.293E-23
  16#3A.B2#E3A
  ```

- ## Characters

  ```
  'A'
  '_'
  '`'
  "This is a string"
  ```

- ## Bits

  ```
  X"FF_AA"
  B"1101_1010"
  O"037"
  ```

Signal Assignment

- ## Delta Delay Model

  ```
  <signal> <= <expression>;
  ```

- ## Inertial Delay Model

  ```
  <signal> <= <expression> AFTER
     <delay>;
  ```

- ## Transport Delay Model

  ```
  <signal> <= TRANSPORT <expression>
     AFTER <delay>;
  ```

- ## Waveforms

  ```
  <signal> <= <value> AFTER <delay>,
     <value> AFTER <delay>, … ;
  ```

Signal Assignment

- ## Delta Delay Model

  ```
  a <= b;
  ```

- ## Inertial Delay Model

  ```
  a <= b AFTER 10 ns;
  ```

- ## Transport Delay Model

  ```
  a <= TRANSPORT b AFTER 10 ns;
  ```

- ## Waveforms

  ```
  a <= b AFTER 10ns, c AFTER 15 ns, … ;
  ```

Conditional Signal Assignment

- Like an if … elsif … else …

```
ARCHITECTURE dataflow OF my_thing IS
BEGIN
   a <= b + e WHEN op = add_em ELSE
        b * d WHEN op = mul_em ELSE
        e + d WHEN sum > 100 ELSE
        b * e;
END dataflow;
```

- Sensitive to all waveforms

- Can include AFTER clause

Selected Signal Assignment

- ## Like a case statement

```
ARCHITECTURE dataflow OF my_thing IS
BEGIN
   WITH opcode SELECT
   a <= b + c WHEN add_em,
        b * c WHEN mul_em,
        b - c WHEN sub_em,
        b / c WHEN div_em,
        UNAFFECTED WHEN OTHERS;
END dataflow;
```

- ## Sensitive to select and all waveforms

- ## Can include AFTER clause

Sub-Programs

- ## Functions

```
FUNCTION <designator> ( < args> )
  RETURN <return-type> IS

  <declarations>

BEGIN

  <statements>

END;
```

- ## Procedures

```
PROCEDURE <identifier> ( < args> ) IS

  <declarations>

BEGIN

  <statements>

END;
```

Function Example

```
function max(L, R: INTEGER) return INTEGER is
begin
    if L > R then
        return L;
    else
        return R;
    end if;
end;
```

File I/O

```
architecture file_io of file_ent is

file DATAINFILE1  : TEXT is in "infile1.dat";
signal datareg1_in: std_logic_vector(15 downto 0);

begin
process(clk, rst)

variable read_in1: std_logic_vector(15 downto 0);
variable n       : line;

begin
   if rst = '1' then
      datareg1_in <= "0000000000000000";
   elsif (clk'event and clk= '1') then
      READLINE(DATAINFILE1, n);
      HREAD(n, read_in1);
      datareg1_in <= read_in1;
end if;
end process;
```

File I/O

```
architecture file_io of file_ent is

file DATAOUTFILE : TEXT is out "outfile.dat";

begin

process(clk, rst)

variable temp_result:std_logic_vector(15 downto 0);
variable r           : line;

begin
   if (clk'event and clk= '1') then
      temp_result :=  datareg3_out;
      hwrite(r, temp_result);
      writeline(DATAOUTFILE, r);
   end if;
end process;

end file_io;
```

Event-Driven Simulation

- Simulator keeps sorted list of future events

- Simulator execution

  - remove next event from list

  - set sim time to time of the event

  - simulate the event (primitives)

  - insert new events (caused by the current event) into the event list

  - run until no more events

Simulation

- Concurrent Signal Assignments
  - Happen in parallel
  - Sensitive to signals in function

- Processes
  - Use sensitivity list
  - OR execute continuously
  - Execute in parallel with CSAs and other processes
  - Occur in 0 simulation time

Signal Assignment

- ## Delta Delay Model

  ```
  a <= b;
  ```

- ## Inertial Delay Model

  ```
  a <= b AFTER 10 ns;
  ```

- ## Transport Delay Model

  ```
  a <= TRANSPORT b AFTER 10 ns;
  ```

- ## Waveforms

  ```
  a <= b AFTER 10ns, c AFTER 15 ns, … ;
  ```

Delta Delays

- Signal assignments with no delay incure a "delta" delay

- Simulation time advances one delta at a time until all deltas for a given time are simulated

- Processes can execute more than once for a given signal at a given time

Delta Delays (example)

```
SIGNAL a, b, c, clk: BIT;
PROCESS (clk)
BEGIN
   a <= b;
   b <= a;
END PROCESS;


PROCESS (a)
   VARIABLE cnt : INTEGER := 0;
BEGIN
   cnt := cnt + 1;
   IF cnt MOD 3 /= 0 THEN
       clk <= NOT clk;
   END IF;
END PROCESS;
```

Data-Flow Models

- ## Normal signal-assignment

```
PROCESS (b, e)
BEGIN
   a <= b + e;
   c <= a * b;
   d <= c * e + b;
END process;

TIME t:        b <= 1         -- stimulus

TIME t+1Δ:    a <= b + e;
               c <= a * b;
               d <= c * e + b;

NOTE: assignments made with OLD value of
   a and c.
```

Data-Flow Models

- ## Concurrent signal-assignment

```
ARCHITECTURE dataflow OF my_thing IS
BEGIN
   a <= b + e;
   c <= a * b;
   d <= c * e + b;
END dataflow;

TIME t:         b <= 1

TIME t+1Δ:    a <= b + e;
              c <= a * b;
              d <= c * e + b;

TIME t+2Δ:    c <= a * b;
              d <= c * e + b;

TIME t+3Δ:    d <= c * e + b;

NOTE: changes to a and c propagate.
```

Inertial Delay Model

- ## Assignment statement:

  ```
  a <= b after 5 ns;
  ```

- ## Effect:

  - 5 ns delay before *a* changes to value of *b*
  - IF pulse on *b* is less than 5 ns wide, a does not change

Transport Delay Model

- Assignment Statement:

```
a <= TRANSPORT b after 5 ns;
```

- Effect:
  - *a* is an exact replica of *b* delayed by 5 ns