Introduction to VHDL

Jiri Gaisler

CTH / Gaisler Research
VHDL history

- VHDL stands for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
- Developed by Intermetrics, IBM and Texas Inst.
- Initial development cost: $33M (1983 level)
- IEEE standard 1076-1987/93
- One of the two most common HDLs
VHDL objectives

- Initial objectives (1983):
  - Modelling of digital systems
  - Un-ambiguous and **executable** specification
  - Co-simulation of models from different companies

- Secondary objectives (1993+):
  - Automatic synthesis
  - Gate-level sign-off simulation
  - RTL sign-off
Language overview

- VHDL is based on Ada semantics
- A program consists of a number of processes
- All processes execute in a virtual environment where a global time and an event queue is maintained
- Time is maintained in discrete steps (e.g. 1 ns) and deltas (e.g. 423 ns, delta 4).
- All inter-process communication occurs through the special data structure *signal*
- The *signal* message delay is at least one *delta*
Graphical VHDL example
Scheduling and execution

- Each process has a list of signals called *sensitivity list* (the list may be empty)

- A process is scheduled for execution when any of the signals in its *sensitivity list* change value, or when a timed suspension expires

- All scheduled processes are executed for one iteration, or until they block on a specific wait statement
Simulation cycle

1. Simulation time is advanced until a signal driver becomes active or a suspended process resumes
2. Signal values are updated from active drivers
3. All processes sensitive to signals with events, and resumed processes, are scheduled for execution
4. All scheduled processes execute until suspension
5. The event queue is updated with new signal driver events
A VHDL entity

- The smallest executable VHDL module is an entity.
- Similar to an Ada task, it consists of an interface declaration and a body.
- It can contain any number of processes and signals.
- It can contain other entities (hierarchical design).
Entity declaration

**Entity** `ram` is

```vhdl
 generic (a : integer);
 port (  
     addr : in integer;
     data : inout integer;
     err  : out integer;
     csn  : boolean -- implicit in
    );

type ....
constant ...
end;
```

- Ports are signals
- Generics are local constants
- Type and constant declarations allowed in entity but less frequently used.
**VHDL Entity structure**

**Entity** `entity_name` **is** -- entity declaration
  generic_declarations;
  port_declarations;
);  

**architecture** `arch_name` **of** `entity_name` **is**
  type_declarations; -- arch. declaration part
  constant_declarations;
  signal_declarations;
  subprogram_declarations;
  component_declarations;

**begin**
  concurrent_statements; -- arch. statement part
  process_statements;
  component_instatiations;
  generate_statements;
  **end**;
Type declarations

- Scalar types
  - Integer, floating-point
  - Enumerated, physical
- Composite
  - Records, arrays
- Access
- File

- Integer: +/- $2^{31}$
- Real: -1E38 to +1E38
- Standard operators defined
  +, -, *, /, <, >, =
- Enum:
  type boolean is (false, true);
  type bit is ('0', '1');
  type short is range 0 to 7;
Type declarations

◆ Physical types

type time is range 0 to 1E18 units
  fs;
  ps = 1000 fs;
  ns = 1000 ps;
  us = 1000 ns;
  ms = 1000 us;
  s = 1000 ms;
end units;

wait for 10 us;
a <= b after (10 ms + 1 ns);

◆ Composite types

type barr is array (1 to 5) of bit;
type v is array (integer range <> ) of real;
subtype farr is x (0 to 127);

signal y : v(10 downto 0);
attributes: v'length, v'range, v'left, v'right

type mrec is record
  a : integer;
end record;

variable x : mrec;
x.a := 3;
Constant declarations

- Constants can occur in entities and sub-programs

```plaintext
constant name : type := value;

constant pi : real := 3.1416;
```

- Aggregates are allowed:

```plaintext
constant v : int_vector (1 to 8) := (1, 2, 3, 4, others => 0);
```

- Constant expression are allowed:

```plaintext
constant v : integer := a**3 + xfunc(34);
```
Signal declarations

- Signal declaration is allowed in the declaration part of an architecture
- An default value is allowed (but not synthesisable)

```vhdl
signal name : type [:= value];

signal s1 : integer;
signal s2 : bit_vector(1 to 2) := “00”; 
```
VHDL signals

- Holds a value of (almost) any data type
- Assignment enters a driver event in the event queue:
  
  ```vhdl
  a <= b after 10 ns; -- event after 10 ns delay
  a <= b;          -- event after 1 delta delay
  ```

- In addition to its current value, several attributes are implicitly maintained: s'event, s'last_event, s'last_value, s'stable ...

- A signal with multiple drivers must have a resolution function declared
Component declarations

- Component declaration is allowed in architecture declaration part and in packages
- Is used to make the interface of an other entity visible
- Must be identical to the original entity, but generics can be removed if not used

```
component adder
  port (  
    a : in integer;  
    b : in integer;  
    c : out integer
  );
end component;
```
Sub-program definition

- Sub-program definition is allowed in architecture declaration part, in sub-program declaration part, and in packages
- Similar semantics to Ada and Pascal

```vhdl
function parity (v : bit_vector) return bit is
  variable b : bit;
begin
  for i in v'range loop
    b := b xor v(i);
  end loop;
  return b;
end;
```
Processes

- Processes are defined through *concurrent statements*, or *process* statements
- Choice of syntax does not change function
- Each concurrent statement is a separate process, with all input signals implicitly added to the sensitivity list
- The process statement allows explicit declaration of sensitivity list, and the use of *wait* statements
- All code within a process statement executes sequentially
Process structure

- Similar structure to a sub-program (procedure)

Label: process [(sensitivity_list)]
  type_declarations;         -- process declaration part
  constant_declarations;
  subprogram_declarations;
  variable_declarations;
begin
  sequential_statements;     -- process statement part
end;

- Wait statement allowed if no sensitivity list

wait for 10 ns;
wait on a, b, c until a > b for 300 ns;
Concurrent vs. Process statements

◆ Concurrent statements

\[ a \leq b + c \text{ after 10 ns; } \]
\[ a \leq b \text{ when inc } = 0 \]
\[ \quad \text{else } b+1 \text{ when inc } =1 \]
\[ \quad \text{else } b+2; \]

◆ Process statement

\[ p1 : \text{process (b, c)} \]
\[ \begin{align*}
    &\begin{align*}
        &\text{begin} \\
        &\quad a \leq b+c \text{ after 10 ns;} \\
        &\quad \text{end process;}
    \end{align*}
\end{align*} \]
\[ p2 : \text{process (b, inc)} \]
\[ \begin{align*}
    &\begin{align*}
        &\text{begin} \\
        &\quad \text{if inc } = 0 \text{ then} \\
        &\quad \quad a \leq b; \\
        &\quad \text{elsif inc } =1 \text{ then} \\
        &\quad \quad a \leq b+1; \\
        &\quad \text{else} \\
        &\quad \quad a\leq b+2; \\
        &\quad \text{end if;} \\
        &\quad \text{end process;}
    \end{align*}
\end{align*} \]
Sensitivity list vs. Wait statement

**Sensitivity list**

p1 : process(a, b, c) begin  
res <= a + b *c;  
end process;

p2 : process(clk) begin  
if clk = '1' then  
res <= a + b*c;  
end if;  
end process;

**Wait statement**

p1 : process begin  
res <= a + b *c;  
wait on a, b, c;  
end process;

p2 : process begin  
res <= a + b*c;  
wait on clk until clk = '1';  
end process;
Concurrent vs. Process statements

- Concurrent statements are limited in complexity, and suitable to express functionality in dataflow manner.
- Process statements allow complex statements such as loops and multi-case selections.
- Procedures and functions must use sequential code.
Component instantiation

- Provides a mean to connect entities in a hierarchy
- The ports transport the signals without delay
- Both named and position signal association allowed
- Outputs can be left open (a=>open)
- Allowed in architecture

architecture rtl of xxx is
signal a, b, c, d, e, f : integer;

component adder
port (a : in integer; b : in integer; c : out integer);
end component;

begin
u0 : adder port map (a => a, b => b, c => c);
u1 : adder port map (d, e, f);
end;
Generate statements

- Allows to include or exclude statements in the architecture
- Loops allowed
- Resolved at elaboration time

Entity adder
generic (fast : boolean);
port map (  
a, b : in bit_vector(7 downto 0);
sum : out bit_vector(7 downto 0);
);

architecture rtl of adder is
begin
a0 : if fast generate
  u0 : csa_adder (a, b, sum);
end generate;
a1 : if not fast generate
  u0 : ripple_adder (a, b, sum);
end generate;
end;

architecture rtl2 of adder is
signal c : bit_vector(8 downto 0);
begin
  c(0) <= '0';
g0 : for i in 0 to 7 generate
  u0 : full_adder port map
      (a(i), b(i), c(i), sum(i), c(i+1));
end generate;
end;
Packages

Similar to Ada, a VHDL package contains a collection of types, constants, component declarations and sub-programs.

Entities or processes cannot be placed in packages!

Usage of packages follows Ada syntax:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith."+";

Package complex_math is
  type complex is record
    r : real;
    i : real;
  end record;

  function cadd (c1, c2 : complex) return complex;
end;

package body complex_math is
  function cadd (c1, c2 : complex) return complex is
    variable res : complex;
    begin
      res.r := c1.r + c2.r;
      res.i := c1.i + c2.i;
      return res;
    end;
end;
```
Pre-defined types and packages

**STANDARD**

package standard is

type boolean is (false, true);
type bit is ('0', '1');
type character is (NUL, SOH, ..... 'A', 'B', 'C' .... DEL);
type integer is range -2**31 to 2**31; -- implementation defined
type real is -1E38 to +1E38; -- implementation defined
type time is ..... 

function now return time;
subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;
type string is array (positive range <>) of character;
type bit_vector is array (natural range <>) of bit;
end standard;
IEEE packages

◆ std_logic_1164

type std_ulogic is ( 
  'U', -- Uninitialized 
  'X', -- Forcing Unknown 
  '0', -- Forcing 0 
  '1', -- Forcing 1 
  'Z', -- High Impedance 
  'W', -- Weak Unknown 
  'L', -- Weak 0 
  'H', -- Weak 1 
  '-' -- Don't care 
);

type std_ulogic_vector is array (natural range <>) of std_ulogic;

Resolved types are called std_logic / std_logic_vectors
IEEE packages

- std_logic_1164, defined functions
- and, nand, or, nor, xor, xnor, not (on scalars and vectors)
- to_bit, to_bitvector, to_stdlogic, to_stdlogicvector
- is_x

- std_logic_arith

type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
type SIGNED is array (NATURAL range <>) of STD_LOGIC;

operators: +, -, *, /, <, >, =, <=, >=, /=, abs, conv_integer, conv_unsigned, conv_signed, conv_std_logic_vector
Examples

- Clock generator

  ```vhdl
  signal clk : std_logic := '0';
  clk <= not clk after 5 ns;
  ```

- Transparent latch

  ```vhdl
  latch : process (clk, d)
  begin
    if clk = '1' then
      q <= d;
    end if;
  end process;
  ```

- flip-flop

  ```vhdl
  dff1 : process (clk)
  begin
    if clk'event and (clk = '1') then
      q <= d;
    end if;
  end process;
  ```

  ```vhdl
  dff2 : process (clk)
  begin
    if rising_edge(clk) then
      q <= d;
    end if;
  end process;
  ```
Library ieee;

use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity async_ram is
    generic ( abits : integer := 10; dbits : integer := 8 );
    port ( address : in std_logic_vector((abits -1) downto 0);
          data : inout std_logic_vector((dbits -1) downto 0);
          csn, wen, oen : in std_logic);
end;

architecture behavioral of async_ram is
    type mem is array(0 to (2**abits -1)) of std_logic_vector((dbits -1) downto 0);
    begin
        r : process(address, data, csn, wen, oen)
        variable memarr : mem;
        variable dout : std_logic_vector((dbits -1) downto 0);
        begin
            if csn = '0' then
                dout := memarr(conv_integer(unsigned(address)));
                if wen'event and (wen = '1') then
                    memarr(conv_integer(unsigned(address))) := data;
                end if;
                if oen = '0' then data <= dout; else data <= (others => 'Z'); end if;
            end if;
        end process;
    end;
Library ieee;
-- 32-bit RAM BANK
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity mem_bank is
  port (     
    address : in std_logic_vector(23 downto 2);    
    data    : inout std_logic_vector(31 downto 0);    
    csn, wen, oen : in std_logic);    
end;

architecture behavioral of async_ram is

component async_ram
  generic ( abits : integer := 10; dbits : integer := 8 );
  port (     
    address : in std_logic_vector((abits -1) downto 0);    
    data    : inout std_logic_vector((dbits -1) downto 0);    
    csn, wen, oen : in std_logic);    
end component;

begin
  mb: for i in 0 to 3 generate
    u0 : async_ram generic map (22, 8);
    port map (address, data(i*8+7 downto i*8), csn, wen, oen);
  end generate
end;
Common problems

- Clock skew
  clkn <= not clk;
  clk2 <= clk;

- Signals are assigned with delay
  a <= b;
  c <= a;

  c is NOT equal to b !!

- Oscillation
  clk <= not clk;

- Inifinite loop (missing wait)
  process
    begin
      statements;
    end process;
Elaboration and start-up

- Before execution starts, the program is elaborated
  - Generates are resolved
  - All signals and variables initialised
  - All processes scheduled for execution
  - Time is set to 0
Execution (simulation)

- The program is run for a given time
- Output can be written to console or files
- Graphical tracing of signals common during debug
- Variables difficult to trace (as in Ada/Pascal)
- No exit() call – difficult to 'stop' a program from within
Execution (simulation)
Not covered topics

- Configurations
- Blocks and guards
- Textio package and file I/O
- Synthesis subset
VHDL resources on the web

◆ Sites

◆ Courses and docs
http://mikro.e-technik.uni-ulm.de/vhdl/vhdl_infos.html
VHDL course modules (3) at rassp.scra.org

◆ References
VHDL Cook-book, Peter Ashenden, Uni. Adelaide