Basic Language Concepts

Describing Design Entities

- Primary programming abstraction is a *design entity*
  - Register, logic block, chip, board, or system
- What aspects of a digital system do we want to describe?
  - Interface: how do we connect to it
  - Function: what does it do?
- VHDL 1993 vs. VHDL 1987
Describing the Interface: The Entity Construct

- The interface is a collection of **ports**
  - Ports are a new programming object: **signal**
  - Ports have a type, e.g., bit
  - Ports have a mode: in, out, inout (bidirectional)

```
entity half_ADDer is
  port ( a, b : in bit;
        sum, carry : out bit );
end entity half_ADDer;
```

The Signal Object Type

- VHDL supports four basic objects: variables, constants, signals and file types (1993)
- Variable and constant types
  - Follow traditional concepts
- The signal object type is motivated by digital system modeling
  - Distinct from variable types in the association of time with values
  - Implementation of a signal is a sequence of time-value pairs!
    - Referred to as the driver for the signal
Example Entity Descriptions

entity ALU32 is
  port (A, B : in bit_vector (31 downto 0);
        Op: in bit_vector (5 downto 0);
        N, Z: out bit);
end entity ALU32;

entity D_ff is
  port (D, Q, Clk, R, S: in bit;
        Q, Qbar : out bit);
end entity D_ff;

Describing Behavior: The Architecture Construct

entity half_adder is
  port (a, b : in bit;
        sum, carry : out bit);
end entity half_adder;

architecture behavioral of half_adder is
begin
  sum <= (a xor b) after 5 ns;
  carry <= (a and b) after 5 ns;
end architecture behavior;

- Description of events on output signals in terms of events on input signals: the _signal assignment_ statement
- Specification of propagation delays
- Type _bit_ is not powerful enough for realistic simulation: use the IEEE 1164 value system
Example Entity Descriptions: IEEE 1164

entity ALU32 is
  port(
    A, B: in std_logic_vector (31 downto 0);
    Op: in std_logic_vector (5 downto 0);
    C: out std_logic_vector (31 downto 0);
    Q, Qbar: out std_logic);
end entity ALU32;

entity D_ff is
  port(
    D, Q, Clk, R, S: in std_logic;
    Q, Qbar: out std_logic);
end entity D_ff;

Describing Behavior: The Architecture Construct

library IEEE;
use IEEE.std_logic_1164.all;

entity half_adder is
  port(a, b : in std_logic;
    sum, carry : out std_logic);
end entity half_adder;

architecture behavioral of half_adder is
begin
  sum <= (a xor b) after 5 ns;
carry <= (a and b) after 5 ns;
end architecture behavioral;

• Use of the IEEE 1164 value system requires inclusion of the library and package declaration statements
Libraries and Packages

- Libraries are logical units that are mapped to physical directories
- Packages are repositories for type definitions, procedures, and functions
  - User defined vs. system packages

![Diagram of package structure]

- package declaration
- specification of the package contents
- package body
- code blocks

Configurations

- Separate the specification of the interface from that of the implementation
  - An entity may have multiple architectures
- Configurations associate an entity with an architecture
  - Binding rules: default and explicit
- Use configurations (more later!)
Design Units

- **Primary design units**
  - Entity
  - Configuration
  - Package Declaration
  - These are not dependent on other design units

- **Secondary design units**
  - Package body
  - Architecture

- Design units are created in design files

- Now you know the layout of a VHDL program!

---

Simple Signal Assignment

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity full_adder is
   port (in1, in2, c_in: in std_ulogic;
         sum, c_out: out std_ulogic);
end entity full_adder;
architecture dataflow of full_adder is
   signal s1, s2, s3 : std_ulogic;
   constant gate_delay: Time := 5 ns;
   begin
      L1: s1 <= (in1 xor in2) after gate_delay;
      L2: s2 <= (c_in and s1) after gate_delay;
      L3: s3 <= (in1 and in2) after gate_delay;
      L4: sum <= (s1 xor c_in) after gate_delay;
      L5: c_out <= (s2 or s3) after gate_delay;
   end architecture dataflow;
```

---

(11) Design Units

(12) Simple Signal Assignment
Simple Signal Assignment Statement

- The constant programming object
  - Values cannot be changed
- Use of signals in the architecture
  - Internal signals connect components
- A statement is executed when an event takes place on a signal in the RHS of an expression
  - 1-1 correspondence between signal assignment statements and signals in the circuit
  - Order of statement execution follows propagation of events in the circuit
  - Textual order does not imply execution order

Implementation of Signals

\[ s \leftarrow (\text{In1 nand In2}) \text{ after gate\_delay} \]
Implementation of Signals (cont.)

• In the absence of initialization, default values are determined by signal type

• Waveform elements describe time-value pairs

• *Transactions* are internal representations of signal value assignments
  – Events correspond to new signal values
  – A transaction may lead to the same signal value

(driver)

Implementation of Signals (cont.)

• *Driver* is set of future signal values: current signal value is provided by the transaction at the head of the list

• We can specify multiple waveform elements in a single assignment statement
  – *Specifying multiple future values for a signal*

• Rules for maintaining the driver
  – *Conflicting transactions*
Example: Waveform Generation

signal <= '0','1' after 10 ns,'0' after 20 ns,'1' after 40 ns;

- Multiple waveform elements can be specified in a single signal assignment statement
- Describe the signal transitions at future point in time
  - Each transition is specified as a waveform element

Resolved Signal Types

- At any point in time what is the value of the bus signal?
- We need to “resolve” the value
  - Take the value at the head of all drivers
  - Select one of the values according to a resolution function
- Predefined IEEE 1164 resolved types are std_logic and std_logic_vector
Conditional Signal Assignment

library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
port (In0, In1, In2, In3 : in std_logic_vector (7 downto 0);
     Sel: in std_logic_vector(1 downto 0);
     Z : out std_logic_vector (7 downto 0));
end entity mux4;
architecture behavioral of mux4 is
begin
    Z <= In0 after 5 ns when Sel = "00" else
         In1 after 5 ns when Sel = "01" else
         In2 after 5 ns when Sel = "10" else
         In3 after 5 ns when Sel = "11" else
         "00000000" after 5 ns;
end architecture behavioral;

• First true conditional expression determines the output value

Unaffected Signals

library IEEE;
use IEEE.std_logic_1164.all;
entity pr_encoder is
port (S0, S1,S2,S3 : in std_logic;
     Z : out std_logic_vector (1 downto 0));
end entity pr_encoder;
architecture behavioral of pr_encoder is
begin
    Z <= "00" after 5 ns when S0 = '1' else
         "01" after 5 ns when S1 = '1' else
         unaffected when S2 = '1' else
         "11" after 5 ns when S3 = '1' else
         "00" after 5 ns;
end architecture behavioral;

• Value of the signal is not changed
• VHDL 1993 only!
### Selected Signal Assignment Statement

library IEEE;
use IEEE.std_logic_1164.all;

entity mux4 is
  port (In0, In1, In2, In3 : in std_logic_vector (7 downto 0);
    Sel: in std_logic_vector(1 downto 0);
    Z : out std_logic_vector (7 downto 0));
end entity mux4;

architecture behavioral of mux4 is
begin
  with Sel select
  Z <= (In0 after 5 ns) when "00",
       (In1 after 5 ns) when "01",
       (In2 after 5 ns) when "10",
       (In3 after 5 ns) when "11",
       (In3 after 5 ns) when others;
end architecture;

- The "when others" clause can be used to ensure that all options are covered
- The "unaffected" clause may also be used here

### A VHDL Model Template

library library-name-1, library-name-2;
use library-name-1.package-name.all;
use library-name-2.package-name.all;

entity entity_name is
  port( input signals : in type;
       output signals : out type);
end entity entity_name;

architecture arch_name of entity_name is
begin
  -- you may have multiple signals of different types
  signal internal-signal-1 : type := initialization;
signal internal-signal-2 : type := initialization;

  -- specify value of each signal as a function of other signals
  internal-signal-1 <= simple, conditional, or selected CSA;
  internal-signal-2 <= simple, conditional, or selected CSA;

  end architecture arch_name;

Delay Models in VHDL

• Inertial delay
  – Default delay model
  – Suitable for modeling delays through devices such as gates
• Transport Delay
  – Model delays through devices with very small inertia, e.g., wires
  – All input events are propagated to output signals
• Delta delay
  – What about models where no propagation delays are specified?
  – Infinitesimally small delay is automatically inserted by the simulator to preserve correct ordering of events

Inertial Delays: Example

- signal <= reject time-expression inertial value-expression after time-expression;
- Most general form of a waveform element
- VHDL 1993 enables specification of pulse rejection width
Transport Delays: Example

architecture transport_delay of half_adder is
signal s1, s2: std_logic := '0';
begin
s1 <= (a xor b) after 2 ns;
s2 <= (a and b) after 2 ns;
sum <= transport s1 after 4 ns;
carry <= transport s2 after 4 ns;
end architecture transport_delay;

Delta Delays: Example

library IEEE;
use IEEE.std_logic_1164.all;
entity combinational is
port (In1, In2: in std_logic;
      z : out std_logic);
end entity combinational;
architecture behavior of combinational is
signal s1, s2, s3, s4: std_logic := '0';
begin
s1 <= not In1;
s2 <= not In2;
s3 <= not (s1 and In2);
s4 <= not (s2 and In1);
z <= not (s3 and s4);
end architecture behavior;
Delta Delays: Behavior

Delay Models: Summary

- Delay models
  - Inertial
    - For devices with inertia such as gates
    - VHDL 1993 supports pulse rejection widths
  - Transport
    - Ensures propagation of all events
    - Typically used to model elements such as wires
  - Delta
    - Automatically inserted to ensure functional correctness of code blocks that do not specify timing
    - Enforces the data dependencies specified in the code
Summary

• Primary unit of abstraction is a design entity

• Design units include
  – Primary design units
    • entity, configuration, package declaration
  – Secondary design units
    • architecture, package body

• Concurrent signal assignment statements
  – Simple, selected, conditional
  – Can be coalesced to form models of combinational circuits