Modeling Constructs

Prof. K. J. Hintz

Department of Electrical
and
Computer Engineering
George Mason University
Modeling Constructs

- **An Entity Declaration**
  - Describes the external interface (I/O signals)

- **Multiple Architecture Bodies**
  - Describe alternative implementations of an entity
    - Increasing levels of fidelity to physical reality
    - Allows simulation to different degrees of detail

- **Hierarchical Approach to Model Building**
Entity Declarations

- Entities are the Primary Hardware Abstraction in VHDL and May Represent
  - Entire system
  - Subsystem
  - Board
  - Chip
  - macro-cell
  - logic gate
Entity Declarations

- May Be Used Directly as a Component in a Design
- May Be the Top Level Module of the Design
  - Reside in library
  - Usable by other designs
- May Have No Port Clause When Used As a Test Bench Since All Signals Are Generated Internally
Entity Declaration Syntax

\texttt{entity identifier is}

\begin{verbatim}
    [ port ( port_interface_list ) ; ]

    { entity_declarative_item }

end [ entity ] [ identifier ] ;
\end{verbatim}
Interface List Syntax

\[
\text{port\_interface\_list} \ <= \n\begin{cases}
\text{( \ identifier \ \ \{ , \ . \ . \ . \} : [ \ mode \] } \\
\text{ subtype\_indication \ [ : = \ expression \ ] )}
\end{cases}
\]

\[
\text{mode} \ <= \n\begin{array}{c}
in \mid \ out \mid \ inout \mid \ buffer \mid \ linkage
\end{array}
\]
Port Clause

Channels for Dynamic Communication Between a Block and Its Environment

Optional

– Not needed for testbench entities, e.g.,

\texttt{entity TestNiCadCharger is}
\texttt{end entity TestNiCadCharger ;}
Port Clause

- **Purpose Is to Define Interface**
  - **in**
    - in only
  - **out**
    - out only
  - **inout**
    - Can be connected to multiple signals
    - Requires bus resolution function
Port Clause

- **buffer**
  - Signal driving buffer drives output
    - *i.e.*, no need to resolve
  - Signal driving buffer can also drive other internal signals
    - *i.e.*, there is an implied buffer isolating internal usage from port signal

- **linkage**
  - Means for connecting to foreign design entities, *e.g.*, Verilog
Port Clause

- Can Be Given Default Value Which Can Be Overridden When Instantiated
  - Unconnected or unassociated signals of mode in must have default expression
entity NiCadCharger is
    port ( Voltage, Current : in real := 0.0 ;
              AC : in bit := '1' ;
              Charged, Recharge: out bit ) ;
end entity NiCadCharger ;
Component vs Entity

- Entities Represent Real Devices Which Have Architectures to Implement Them
- Components Represent Interfaces to Entities or Virtual Devices
  - Need to be declared
    - may be instantiated at same time
    - can be used to build structural model, instantiated later
Component Declaration

- Declares a Virtual Design Entity Interface
- May Be Used to Associate a Component Instance With a Design Entity in a Library by Using
  - Component configuration
  - Configuration specification
Component Declaration Syntax

```
component identifier [ is ]
    [ generic_clause ]
    [ port_clause ]
end component [ identifier ] ;

generic_clause <=
    generic ( generic_interface_list ) ;
```
Component Declaration, \textit{e.g.}


class component \textbf{mux}_2\_to\_1 \textbf{is}

\begin{verbatim}
generic ( tpd : time );
port ( in_1, in_2, mux_out );
end component mux;
\end{verbatim}


Component Instantiation Syntax

Identifier: [ component ]

component_identifier

[ generic map ( generic_association_list )]

[ port map ( port_association_list ) ];
Direct Component Instantiation

Identifier: \texttt{entity}

entity\_identifier

\[
[ ( \text{architecture\_identifier} ) ]
\]

\[
[ \text{port map} ( \text{port\_association\_list} ) ] ;
\]
Direct Component Instantiation, e.g.

And2_1: entity
And2

[ ( TI74LS00 ) ]
[ port map ( pin2 => in_1,
in3 => in_2,
in1 => out ) ] ;
Configuration Declaration

- Declares Virtual Design Entity That May Be Used for Component Instantiation
- Components Need Not Be Bound When Declared, but Specified in Configuration
- Binds a Component Instance With a Design Entity in a Library
Basic Configuration Syntax

configuration identifier of entity_name is

    { configuration_declarative_item }

block_configuration

end [ configuration ] [ identifier ] ;

where identifier is the entity at the top of the design hierarchy
Configuration Declarative Syntax

configuration_declarative_item  <=
     use_clause
     attribute_specification
     group_declaration

(a group_declaration declares a named collection of named entities)
Block Configuration Syntax

- A Block is an Identified Group of Concurrent Statements

```
block_configuration <=

  for block_specification
  {  use_clause  }
  {  configuration_item  }
end for ;
```

Block Specification Syntax

block_specification  <=

architecture_name
|  block_statement_label
|  generate_statement_label
[  (  index_specification  )  ]
Configuration Item Syntax

configuration_item  <=

  block_configuration
  |  component_configuration
Component Configuration Syntax

- Associates Binding Information With Component Labels

\[
\text{for } \text{component\_specification} \\
\quad [ \text{binding\_indication} ] \\
\quad [ \text{block\_configuration} ] \\
\text{end for} ;
\]


Binding Indication

\[
\text{binding\_indication} \leq \]

\[
\begin{array}{l}
\text{[ use entity\_aspect ]} \\
\text{[ generic\_map\_aspect ]} \\
\text{[ port\_map\_aspect ]}
\end{array}
\]
Entities/Components

- Entities Can Be Defined Locally or in a Library
- Components Are Local Linkages to Entities in Libraries
- Either Can Be Used in an Architecture, but Component Is More General and Allows for Reuse of Entities
Architecture Bodies

There May Be Multiple Architecture Bodies of the Same Entity With Each Architecture Body Describing a Different Implementation of the Entity.

- Behavior using the sequential or concurrent statements
- Structure of the entity as a hierarchy of entities
Architecture Bodies

- Declarations Define Items That Will Be Used to Construct the Design Description.
  - Signals used to connect submodules in a design
  - Component port declarations are signals within the entity
architecture identifier of entity_name is
{ block_declarative_item }
begin
{ concurrent_statement }
end [ architecture ] [ identifier ];
Concurrent Statements

- No Temporal Ordering Among the Statements

- A Signal Assignment Statement Is a Sequential Statement and Can Only Appear Within a Process.
Concurrent Statements

- A Process of Sequential Statements Behaves As If It Were One Concurrent Statement
  - Internally sequential
  - Externally concurrent
Signal Declarations

- Signals Can Be Declared Internal to an Architecture to Connect Entities

- Variables Are Not Appropriate Since They Do Not Have the Temporal Characteristics of Hardware
Signal Declarations

- Signals Declared Within an Entity Are Not Available Unless Specified in the Port Clause of the Entity Declaration.

- Discrete Event Simulation
  - Signal changes are scheduled in the future
Signal Syntax

```
signal identifier { , . . . } :
    subtype_indication [ := expression ] ;

    [ label : ] name <= [ delay_mechanism ]
waveform ;
```
Waveform Syntax

waveform <=

( value_expression [ after time_expression ] )

{ , . . . }

Copyright © 1997/8/9, KJH, 545_6, 5/7/2001
Signal Assignment, *e.g.*, Recharge <= `1` *after* 5 ns, `0` *after* 2005 ns;

- Executes in Zero Time
- Schedules Future Event
- Time References Are Relative to Current Time
  - *Recharge* returns to zero 2 ms after it goes high, not 2.005 ms after it goes high
Discrete Event Simulation

- **Transaction**
  - A scheduled change in a signal value

- **Active Signal**
  - A simulation cycle during which a signal transaction occurs

- **Event**
  - A transaction which results in a signal’s value changing
Signal Attributes

Given a signal, $S$, and a value $T$ of type time

$S$’delayed $(T)$ A signal that takes on the same values as $S$ but is delayed by time $T$

$S$’stable $(T)$ A Boolean signal that is true if there has been no event on $S$ in the time interval $T$ up to the current time, else false

$S$’quiet $(T)$ A Boolean signal that is true if there has been no transaction on $S$ in the time interval $T$ up to the current time, else false
Signal Attributes

- **S’transaction**: A signal of type bit that changes value from ‘0’ to ‘1’ or vice versa each time there is a transaction on S.
- **S’event**: True if there is an event on S in the current simulation cycle, else false.
- **S’active**: True if there is a transaction on S in the current simulation cycle, else false.
- **S’last_event**: The time interval since the last event on S.
- **S’last_active**: The time interval since the last trans. on S.
- **S’last_value**: Value of S just before the last event on S.
Signal Attributes, \textit{e.g.}, 1*

\begin{verbatim}
if clk'event
    and ( clk = '1' \textbf{or} clk = 'H' )
    and ( clk'last_value = '0' \textbf{or}
        clk'last_value = 'L')
then
    assert d'last_event >= Tsu
    report "Timing error: d changed within setup time of clk" ;
end if ;
\end{verbatim}

\footnote{Ashenden, p112}
Signal Attributes, e.g., 2*

entity edge_triggered_Dff is
  port ( D, clk, clr : in bit ;
        Q : out bit )
end entity edge_triggered_Dff ;
architecture behavioral of edge_triggered_Dff is
begin
  state_change : process ( clk, clr ) is

*Ashenden, p113
begin
  if clr = '1' then
    Q <= '0' after 2 ns ;
  elsif clk'event and clk = '1' then
    Q <= D after 2 ns ;
  end if ;
end process state_change ;
end architecture behavioral ;
Wait Statements

- Until Statements Depend on Time
- Wait Statements Depend on Signals

```
[ label : ] wait [ on signal_name
    { , . . . } ]
    [ until Boolean_expression ]
    [ for time_expression ]
```

Sensitivity Clause
Condition Clause
Timeout Clause
Wait Statements

- All Clauses Are Optional

- Wait Statements Are The Only Ones That Take More Than Zero Time to Execute
Wait Statements

- Simulation Time Only Advances With Wait Statements

- A Process Without a Wait Statement Is an Infinite Loop.
  - Processes with sensitivity lists have an implied wait statement at the end of the process.
  - A process must have either a wait statement or a sensitivity list.
Sensitivity Clause

- A list of signals to which the process responds
- An event (change in value) on any signal causes the process to resume
- Shorthand—sensitivity list in heading rather than wait statement at end of process.
Sensitivity Clause, \textit{e.g.}

\begin{verbatim}
process (clk, clr) is
begin
  wait on clk, clr;
end;
\end{verbatim}

\begin{verbatim}
process is
begin
  wait on clk, clr;
end;
\end{verbatim}
Condition Clause

- When Condition Is True, Process Resumes
- Condition Is Tested Only While the Process Is Suspended
  - Even if the condition is true when the wait is executed, the process will suspend
  - In order to test the condition,
    » A signal in the sensitivity list must change, or,
    » *IFF* there is no sensitivity list, an event must occur on a signal within the condition
Timeout Clause

- Specifies the Maximum Simulation Time to Wait.

- A Sensitivity or Condition Clause May Cause the Process to Resume Earlier.
Concurrent Signal Assignments

- Functional Modeling Implements Simple Combinational Logic.

- Concurrent Signal Assignment Statements Are an Abbreviated Form of Processes
  - Conditional signal assignment statements
  - Selected Signal Assignment Statements
Conditional Signal Assignments

- Shorthand *if* Statement

- Sensitive to ALL Signals Mentioned in Waveforms and Conditions.
Conditional Signal Assignment

Syntax

```
[ label : ] name <= [ delay_mechanism ]
{ waveform when Boolean_expression else }
waveform [ when Boolean_expression ] ;
```
Can make Waveform Not Schedule a Transaction on the Signal in Response to an Event using unaffected

Can Only Be Used in Concurrent Signal Assignment Statements

\[\text{when not priority\_waiting and server\_status = ready else unaffected ;}\]
Selected Signal Assignments

■ Shorthand for `case` Statement

■ All Rules of `case` Statement Apply

■ `unaffected` Also Applies
Selected Signal Assignments

[ label : ] with expression select
name <= [ delay_mechanism ]
{ waveform when choices , }
waveform when choices ;
Concurrent Assertions

- Shorthand for Process With a Sequential Assertion Statement
- Checks $Boolean_{expression}$ Each Time Signal Mentioned in It Changes

[ label : ] assert $Boolean_{expression}$
[ report expression ]
[ severity expression ] ;
Concurrent Assertions

- Compact Manner for Including Timing and Correctness Checks in Behavioral Models
  - e.g., for S-R flip flop

```python
assert not ( s = '1' and r = '1' )
report "Illegal inputs" ;
```
Entity and Passive Processes

entity identifier is

[ port ( port_interface_list ) ; ]

{ entity_declarative_item }

[ begin

{ concurrent_assertion_statement

| passive_concurrent_procedure_call_statement

| passive_process_statement } ]

end [ entity ] [ identifier ] ;
Passive Statements

- Statements Are Passive If They Do Not Affect the Operation of the Entity in Any Way.

- Concurrent Assertion Statements Are Passive Since They Only Test Conditions
Passive Statements

- A Process Statement Is Passive If It Does NOT Contain
  - any signal assignments, or
  - calls to procedures containing signal assignment statements

- Concurrent Procedure Call Statements
  - not yet covered
Passive Statement Example*

entity S_R_flipflop is
  port ( s , r : in bit ;
        q , q_n : out bit ) ;
begin
  check:  assert not ( s = '1' and r = '1' )
  report "Incorrect use of S_R_flip_flop: s and r both '1'" ;
end entity S_R_flipflop ;
End of Lecture