CPE/EE 422/522
Advanced Logic Design
L11
Electrical and Computer Engineering
University of Alabama in Huntsville

Outline

- What we know
  - How to model Combinational Networks in VHDL
    - Structural, Dataflow, Behavioral
  - How to model Flip-flops in VHDL
  - Processes
  - Delays (delta, transport, inertial)
  - How to model FSM in VHDL
  - Wait statements
  - Variables, Signals, Arrays
- What we do not know
  - VHDL Operators
  - Procedures, Functions
  - Packages, Libraries
  - Additional Topics (if time)

Review: VHDL Operators

1. Binary logical operators: and or nand nor xor xnor
2. Relational: /= <= <= >
3. Shift: sll srl sla sra rol ror
4. Adding: + - & (concatenation)
5. Unary sign: + -
6. Multiplying: * / mod rem
7. Miscellaneous: not abs **

- Class 7 has the highest precedence (applied first), followed by class 6, then class 5, etc

Example of VHDL Operators

In the following expression, A, B, C, and D are bit_vectors:

(A and B or C or D) = "100101"

The operators would be applied in the order:

not, &

If A = "1111", B = "1111", C = "01000" , and D = "110111", the computation would proceed as follows:

not A = "0000" (bit-by-bit complement)
A and B = "11000" (concatenation)
C or D = "110111" (rotate right 1 places)
A or (not B or C or D) = "111011" (bit-by-bit and)
(A and B or C or D) = "110100" (bit-by-bit and)

VHDL Functions

- Functions execute a sequential algorithm and return a single value to calling program

function rotate_right (arg: bit_vector)
return bit_vector is
begin
return arg xor arg
end rotate_right;

A = "1001011"
B <= rotate_right(A);

General form

function function-name (formal-parameter-list)
return return-type is
declarations;
begin
sequential statements -- must include return-value;
end function-name;
For Loops

General form of a for loop:

```vhdl
for loop-index in range loop
    sequential statements
end loop
```

For Loop Example:

```vhdl
for i in 1 to 5 loop
    expression
end loop:
```

Add Function

This function adds 2 4-bit vectors and a carry:

```vhdl
function add(A: bit_vector(3 downto 0); carry: bit) return bit_vector(3 downto 0);
```

VHDL Procedures

• Facilitate decomposition of VHDL code into modules
• Procedures can return any number of values using output parameters

General form:

```vhdl
procedure procedure_name (formal-parameter-list) is
[declarations]
begin
    Sequential-statements
end procedure_name;
```

Procedure for Adding Bit_vectors

This procedure adds two n-bit bit vectors and a carry and returns the n-bit sum and a carry. Add1 and Add2 are assumed to be of the same length and dimension n in 0-downto-0.

```vhdl
procedure Add; (Add1, Add2, in bit_vector;
    C: in bit;
    Sum: out bit_vector;
    Cout: out bit;
begin
    C := C;
    for i in 0 to n-1 loop
        Sum(i) <= Add1(i) or Add2(i) or (Add1(i) and C);
        Cout <= (Add1(i) and Add2(i)) or (Add1(i) and C) or (Add2(i) and C);
    end loop;
end add;
```

Parameters for Subprogram Calls

<table>
<thead>
<tr>
<th>Mode</th>
<th>Class</th>
<th>Procedure Call</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>constant</td>
<td>expression</td>
<td>expression</td>
</tr>
<tr>
<td></td>
<td>signal</td>
<td>signal</td>
<td>signal</td>
</tr>
<tr>
<td>out/inout</td>
<td>variable</td>
<td>variable</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>signal</td>
<td>signal</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>variable</td>
<td>variable</td>
<td>n/a</td>
</tr>
</tbody>
</table>

1 default for functions
2 default for in mode
3 default for out/in mode

Packages and Libraries

• Provide a convenient way of referencing frequently used functions and components

Package declaration

```vhdl
package package-name is
    package declarations;
end (package);[package-name];
```

Package body [optional]

```vhdl
package body package-name is
    package body declarations;
end (package);[package-name];
```
VHDL Model for a 74163 Counter

- 74163 – 4-bit fully synchronous binary counter
- Counter operations

<table>
<thead>
<tr>
<th>Control Signals</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Q3, Q2, Q1, Q0</td>
</tr>
<tr>
<td>x</td>
<td>0, 0, 0, 0</td>
</tr>
<tr>
<td>1</td>
<td>0, 1, 0, 0</td>
</tr>
<tr>
<td>1</td>
<td>1, 0, 1, 0</td>
</tr>
</tbody>
</table>

- Generate a Cout in state 15 if T=1
- Cout = Q3Q2Q1Q0

Cascaded Counters
Signal Attributes

Attributes associated with signals that return a value

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>A'event</td>
<td>True if an event occurred during the current data else false</td>
<td></td>
</tr>
<tr>
<td>A'active</td>
<td>True if a transaction occurred during the current data else false</td>
<td></td>
</tr>
<tr>
<td>S'LAST_EVENT</td>
<td>Time elapsed since the previous event on S</td>
<td></td>
</tr>
<tr>
<td>S'LAST_VALUE</td>
<td>Value of S before the previous event on S</td>
<td></td>
</tr>
<tr>
<td>S'LAST_ACTIVE</td>
<td>Time elapsed since previous transaction on S</td>
<td></td>
</tr>
</tbody>
</table>

S'event – true if a change in S has just occurred
S'active – true if A has just been reevaluated, even if A does not change

Signal Attributes (cont’d)

entity test is
begin
A <= not A after 20 ns;
B <= '1';
C <= A and B;
end;
end bmtest;

ns /test/a   /test/line__15/bev
delta /test/b   /test/line__15/bac
ns /test/l15/aw /test/line__15/cev
ns /test/l15/aw /test/line__15/ncv
0 +0 0 0 0 0
0 +1 0 1 0 0
20 +1 1 1 0 0
40 +0 0 1 1 1
40 +1 0 1 0 0

Signal Attributes (cont’d)

Additional Topics in VHDL

- Attributes
- Transport and Inertial Delays
- Operator Overloading
- Multivalued Logic and Signal Resolution
- IEEE 1164 Standard Logic
- Generics
- Generate Statements
- Synthesis of VHDL Code
- Synthesis Examples
- Files and Text IO
Using Attributes for Error Checking

```vhdl
check: process
begin
  wait until rising_edge(Clk);
  assert (p'stable(setup_time))
  report("Setup time violation")
  severity error;
  wait for hold_time;
  assert (p'stable(hold_time))
  report("Hold time violation")
  severity error;
end process check;
```

Array Attributes

A can be either an array name or an array type.

Array attributes work with signals, variables, and constants.

Recap: Adding Vectors

-- This procedure adds two n-bit vectors and a carry and
-- results are n-bit sum and a carry. Add1 and Add2 are assumed
-- to be of the same length and dimension n-1 down to 0.

```vhdl
procedure Add1(A1: in bit_vector; B1: in bit; C: in bit; signal Sum: out bit_vector; signal Carry: out bit)
begin
  C := 0;
  for i in 0 to n-1 loop
    Sum(i) := A1(i) xor B1 xor C;
    C := (A1(i) and B1) or (A1(i) and C) or (B1 and C);
  end loop;
end Add1;
```

Use attributes to write more general procedure that places
no restrictions on the range of vectors other than the lengths must be same.

Procedure for Adding Bit Vectors

-- This procedure adds two bit vectors and a carry and returns a sum.
-- And a carry. Both bit vectors should be of the same length.

```vhdl
procedure Add2(A2: in bit_vector; B2: in bit; signal Sum: out bit_vector; signal Carry: out bit)
begin
  for i in 0 to n-1 loop
    Sum(i) := A2(i) xor B2 xor Carry;
    Carry := (A2(i) and B2) or (A2(i) and Carry) or (B2 and Carry);
  end loop;
end Add2;
```
Transport and Inertial Delay (cont’d)

\[ Z_3 = \text{reject 4 ns } X \text{ after 10 ns}; \]
\[ Z_2 = \text{after 6 ns}; \]
\[ Z_1 = \text{transport } \]
\[ A = \text{transport } B \text{ after 1 ns}; \]
\[ A = \text{transport } C \text{ after 2 ns}; \]

Statements executed at time \( T \)

- \( B \text{ at } T+1, C \text{ at } T+2 \)
- \( A \leq B \text{ after 1 ns}; A \leq C \text{ after 2 ns}; \)

Statements executed at time \( T \)

- \( C \text{ at } T+1 \)
- \( A \leq B \text{ after 1 ns}; A \leq C \text{ after 2 ns}; \)

Statements executed at time \( T \)

- \( C \text{ at } T \)
- \( A \leq B \text{ after 2 ns}; A \leq C \text{ after 1 ns}; \)

---

Transport and Inertial Delay

\[ Z_1 = \text{transport } \]
\[ Z_2 = \text{after 6 ns}; \]
\[ Z_3 = \text{reject 4 ns } X \text{ after 10 ns}; \]

---

Operator Overloading

- Operators +, - operate on integers
- Write procedures for bit vector addition/subtraction
  - addvec, subvec
- Operator overloading allows using + operator to implicitly call an appropriate addition function
- How does it work?
  - When compiler encounters a function declaration in which the function name is an operator enclosed in double quotes, the compiler treats the function as an operator overloading ("+")
  - When a "+" operator is encountered, the compiler automatically checks the types of operands and calls appropriate functions

---

VHDL Package with Overloaded Operators

- This package provides two overloaded functions for the plus operator
  - package bit_xor is
    - function "+" (bit_vector, bit_vector): return bit_vector;
    - end bit_xor;
  - library stdlib;
  - entity bit_xor is
    - function "+" (bit_vector, bit_vector): return bit_vector;
    - end bit_xor;
  - package body bit_xor is
    - function "+" (bit_vector, bit_vector): return bit_vector;
    - begin
      for i in bit_vector range loop
        return (bit_vector(1) and bit_vector(2)) or (bit_vector(1) and bit_vector(3)) or (bit_vector(2) and bit_vector(3));
      end loop;
    - end bit_xor;
  - end package;

---

Overloaded Operators

- A, B, C – bit vectors
- A <= B + C + 3 ?
- A <= 3 + B + C ?

- Overloading can also be applied to procedures and functions
  - procedures have the same name – type of the actual parameters in the procedure call determines which version of the procedure is called

---

Multivalued Logic

- Bit (0, 1)
- Tristate buffers and buses => high impedance state 'Z'
- Unknown state 'X'
  - e.g., a gate is driven by 'Z', output is unknown
  - a signal is simultaneously driven by '0' and '1'

---
Tristate Buffers

Resolution function to determine the actual value of f since it is driven from two different sources.

Signal Resolution

- VHDL signals may either be resolved or unresolved.
- Resolved signals have an associated resolution function.
- Bit type is unresolved — there is no resolution function.
- If you drive a bit signal to two different values in two concurrent statements, the compiler will generate an error.

Signal Resolution (cont'd)

Signal R : X01Z := 'Z'; ...

 Resolution function

 AND and OR Functions Using X01Z

AND and OR Functions Using X01Z

IEEE 1164 Standard Logic

- 9-valued logic system
  - 'U' — Uninitialized
  - 'X' — Forcing Unknown
  - '0' — Forcing 0
  - '1' — Forcing 1
  - 'Z' — High impedance
  - 'W' — Weak unknown
  - 'L' — Weak 0
  - 'H' — Weak 1
  - 'C' — Don't care
  - 'u' — Forcing and weak signal are tied together, the forcing signal dominates.
  - 'z' — Useful in modeling the internal operation of certain types of ICs.

In this course we use a subset of the IEEE values: X10Z.
Resolution Function for IEEE 9-valued

```haskell
function 'and' (a : std_logic; b : std_logic) return std_logic begin
    return (a_and_b, "AND Function for std_logic_vectors")
end;
```

AND Table for IEEE 9-valued

```haskell
function 'and' (a : std_logic; b : std_logic) return std_logic begin
    return (a_and_b, "AND Function for std_logic_vectors")
end;
```

AND Function for std_logic_vectors

```haskell
function 'and' (a : std_logic; b : std_logic) return std_logic begin
    return (a_and_b, "AND Function for std_logic_vectors")
end;
```