CPE/EE 422/522
Advanced Logic Design
L12
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)

LAB 4: Keypad Scanner

• Lab4 preparation material
• Telephone keypad scanner
  – Section 3.5 in the textbook
  – Implemented using PLD (not relevant for you)

LAB 4: Block Diagram

• Keypad is wired in matrix form
  – switches are at the intersections of rows and columns
• Assumption: only one key is pressed at time
• \( N = N_3N_2N_1N_0 \)
  – 0 – 0000
  – 1 – 1001
  – * - 1010
  – # - 1011
• \( V = 1 \): when valid key is detected it is active for one clock cycle time
LAB 4: Scan Procedure

1. Apply logic 1s to columns C0, C1, C2 and wait
2. If any key is pressed
   a 1 will appear on R0, R1, R2, or R3
3. Apply 1 to column C0 only;
   if any of RI's is 1, a valid key is detected;
   set V=1 and corresponding N
4. If no key is detected in column C0 apply 1 on C1;
   Repeat the same for C2
5. When a valid key is detected, apply 1s to C0, C1, C2 and wait until no key is pressed
   • ensure that only one valid signal is generated each time a key is pressed

LAB 4: Debouncing

• Problem: with mechanical switch the contact will bounce causing noise in the switch output
  • contact may bounce for several milliseconds

  ![Diagram of Mechanical Switch with Bounce]

• Solution: after a switch closure has been detected, wait for bounce to settle down before reading the key

LAB 4: Debouncing and Synchronization Circuit

• Proposed debouncing circuit
• Important: clock cycle time must be greater than the bounce time

LAB 4: Scanner Modules

![Diagram of Scanner Modules]
LAB 4: Scanner

• Problem: what is Kd in S5 if we have a key pressed in column C2?

• Solution

LAB 4: Decoder

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

  function rotate_right (reg: bit_vector) return bit_vector is
  begin
    return reg(1 downto 0);
  end rotate_right;

• General form

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

Review: VHDL Functions

Review: For Loops
Review: 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;
  ```
  
  ```vhdl
  procedure_name (actual-parameter-list);
  ```

Review: Parameters for Subprogram Calls

<table>
<thead>
<tr>
<th>Mode</th>
<th>Class</th>
<th>Actual Parameter</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>constant</td>
<td>expression</td>
<td>expression</td>
</tr>
<tr>
<td>out/inout</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 mode for functions
2. default for in mode
3. default for out/inout 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-name;
  ```

- Package body [optional]
  ```vhdl
  package body package-name is
  package body declarations
  end [package body][package-name];
  ```

Library BITLIB – bit_pack package

```vhdl
package BITLIB is
  constant vec := ("0", "1");
  function vec2bin (vec : in string) return string;
  function bin2vec (bin : in string) return string;
end package;
```

```vhdl
library BITLIB;
```

```vhdl
entity library_entity is
  port (input signal vec);
end entity;
```

```vhdl
architecture structural of library_entity is
  begin
    process (input)
    variable vec : vec;
    variable bin : string;
    begin
      vec := vec2bin(vec);
      bin := bin2vec(bin);
      vec := vec2bin(vec);
      bin := bin2vec(bin);
    end process;
end architecture;
```
Library BITLIB – bit_pack package

```vhdl
package body bit_pack is
  -- The function s.bitizes 2-bit numbers, making a 5-bit array.
  function s.bitizes (x : in    bit_vector) return (s.bitizes) is
  begin
    return (s.bitizes);  
  end s.bitizes;

  variable s.bitizes : bit_vector;
  variable s.bitizes : bit_vector;

end bit_pack;
```

Library BITLIB – bit_pack package

```vhdl
architecture coin of hcolbe is
begin
  A <= A and A2 and B'after (DELAY);
end coin;
```

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

Signal Attributes

Attributes associated with signals that return a value

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>S EVENT</td>
<td>True if an event occurred during the current clock cycle, else false</td>
</tr>
<tr>
<td>S ACTIVE</td>
<td>True if a transition occurred during the current clock cycle, else false</td>
</tr>
<tr>
<td>S LAST EVENT</td>
<td>Time elapsed since the previous event on S</td>
</tr>
<tr>
<td>S LAST VALUE</td>
<td>Value of S before the previous event on S</td>
</tr>
<tr>
<td>S LAST ACTIVE</td>
<td>Time elapsed since previous transition on S</td>
</tr>
</tbody>
</table>

A event – true if a change in S has just occurred

A active – true if A has just been reevaluated, even if A does not change
Signal Attributes (cont’d)

- Event
  - occurs on a signal every time it is changed

- Transaction
  - occurs on a signal every time it is evaluated

Example:

A <= B   - B changes at time T

<table>
<thead>
<tr>
<th>Event</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Signal Attributes (cont’d)

entity test is
architecture bmtest of test is
  begin
    A <= not A after 20 ns;
    B <= '1';
    C <= A and B;
    process(A, B, C)
      variable Aev : bit;
      variable Aac : bit;
      variable Bev : bit;
      variable Bac : bit;
      variable Cev : bit;
      variable Cac : bit;
    begin
      if (A’event) then Aev := '1';
      else Aev := '0';
    end if;
    if (A’active) then Aac := '1';
    else Aac := '0';
    end if;
    if (B’event) then Bev := '1';
    else Bev := '0';
    end if;
    if (B’active) then Bac := '1';
    else Bac := '0';
    end if;
    if (C’event) then Cev := '1';
    else Cev := '0';
    end if;
    if (C’active) then Cac := '1';
    else Cac := '0';
    end if;
    end process;
  end bmtest;
Examples of Signal Attributes

-** VHDL Code for Attribute Test**

entity attr_pv is
  port (Clk: in bit);
end attr_pv;

architecture test of attr_pv is
  signal A, C:delayed(A); B:trans: bit;
begin
  A <= 0 and C;
  B <= A; C <= delayed(A); A <= delayed(A);
end test;

Using Attributes for Error Checking

-** Assert Statement**

  - If boolean expression is false
  - display the string expression on the monitor
  - Severity levels: Note, Warning, Error, Failure

-** Array Attributes**

  - An array attribute can be either an array name or an array type.
  - Array attributes work with signals, variables, and constants.

-** Examples**

  - WAVR: ROM is array (i) to 15.7 [ns] (of clk freq)
  - Signal ROM: ROM

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Returns</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="i">LW</a></td>
<td>left boundary of i'th index range</td>
<td>ROM1(i) &lt;= 9</td>
</tr>
<tr>
<td><a href="i">RW</a></td>
<td>right boundary of i'th index range</td>
<td>ROM1[i+2] &lt;= 10</td>
</tr>
<tr>
<td><a href="i">H</a></td>
<td>highest boundary of i'th index range</td>
<td>ROM1(i) &lt;= 15</td>
</tr>
<tr>
<td><a href="i">L</a></td>
<td>lowest boundary of i'th index range</td>
<td>ROM1[i+2] &lt;= 7</td>
</tr>
<tr>
<td><a href="i">RANGE</a></td>
<td>i'th index range</td>
<td>ROM1(i) &lt;= 0 to 15</td>
</tr>
<tr>
<td><a href="i">REVERSE</a></td>
<td>i'th index range reversed</td>
<td>ROM1(REVERSE) &lt;= 15</td>
</tr>
<tr>
<td><a href="i">LENGTH</a></td>
<td>size of i'th index range</td>
<td>ROM1(LENGTH(i)) &lt;= 34</td>
</tr>
</tbody>
</table>

  A can be either an array name or an array type.
Recap: Adding Vectors

This procedure adds two n-bit vectors and a carry and returns an n-bit sum and a carry. Add1 and Add2 are assumed to be of the same length and dimensioned N-1 downto 0.

```pascal
procedure Add1;  
begin  
  for i from N-1 downto 0 do  
    Sum[i] := Add1[i] xor Add2[i] xor C[i];  
  C[N-1] := (Add1[0] and Add2[0]) or (Add1[0] and C[0]) or (Add2[0] and C[0]);  
end;  
```

Note: Add1 and Add2 vectors must be dimensioned as N-1 downto 0.

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.

```pascal
procedure Add2;  
begin  
  C := C[0];  
  for i from 0 to N-1 do  
    Sum[i] := Add1[i] xor Add2[i] xor C[i];  
  C[N] := (Add1[0] and Add2[0]) or (Add1[0] and C[0]) or (Add2[0] and C[0]);  
end;  
```

Transport and Inertial Delay

Transport and Inertial Delay (cont’d)

23 <= reject 4 ns X after 10 ns;
Reject is equivalent to a combination of inertial and transport delay:
2m <= X after 4 ns;
23 <= transport 2m after 6 ns;

Statements executed at time T
- B at T+1, C at T+2
  A <= transport B after 1 ns;
  A <= transport C after 2 ns;

Statements executed at time T
- C at T+2:
  A <= B after 1 ns;  
  A <= transport B after 2 ns;
  A <= C after 2 ns;  
  A <= transport C after 1 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
library IEEE;
package addvec is
  function add (a1, a2 : bit_vector) return bit_vector;
end addvec;
library IEEE;
package body addvec is
  function add (a1, a2 : bit_vector) return bit_vector is
    variable c : bit := '0';
    variable sum : bit_vector; length := a1'length - 1;
    begin
    loop
      if a1'length = a2'length then
        sum (length) := a1 (length) or a2 (length) and c;
      else
        sum (length) := 'x';
      end if;
      c := a1 (length) and a2 (length) or (c and '1');
      loop
        return (sum);
      end loop;
    end loop;
  end add;
end body addvec;

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)

```vhdl
signal R : X01Z := 'Z'; ...
R <= transport '0' after 2 ns, 'Z' after 6 ns;
R <= transport '1' after 4 ns;
R <= transport '1' after 8 ns, '0' after 10 ns;
```

Resolution Function for X01Z

```
package p004_dr in
    type n_0102 is ('0', '1', 'Z', 'Y');
    type n_0102_vector is array (natural range <>) of n_0102;
end p004_dr;
package body p004_dr is
    function resolved (n_0102_vector) return n_0102_vector is
        begin
            if (length = 1) then
                return (n_0102_vector); end if;
            for i in n_0102_vector'range loop
                result (n_0102_vector'i) := resolved ((n_0102_vector'i, n_0102_vector'i));
            end loop;
            return result;
        end function;
end p004_dr;
```
### AND and OR Functions Using X01Z

<table>
<thead>
<tr>
<th>AND</th>
<th>X</th>
<th>0</th>
<th>1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>X</th>
<th>0</th>
<th>1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### 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
  - '-' – Don’t care

If forcing and weak signal are tied together, the forcing signal dominates.

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
CONSTANT resolution_table : std_logic_table => ( )
-- | U | X | 0 | 1 | 2 | W | L | H |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
```

### AND Table for IEEE 9-valued

```haskell
CONSTANT and_table : std_logic_table => ( )
-- | U | X | 0 | 1 | 2 | W | L | H |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
```

AND Function for std_logic_vectors

```vhdl
function "and" ( l : std_logic_vector; r : std_logic_vector ) return std_logic_vector is
begin
    return (l & r);
end "and";
```

```vhdl
function "and" ( l : std_logic_vector; r : std_logic_vector ) return std_logic_vector is
begin
    variable result : std_logic_vector ( l'length ) := (others => '0');
    variable r : std_logic_vector ( r'length ) := r;
    for i in result'range loop
        result(i) := l(i) and r;
    end loop;
    return result;
end "and";
```