CS/EE 3710: Computer Design Laboratory

Lecture 6: Subprograms, Packages, and Libraries

(Chapter 6: YalamANCHILI)

Fall Semester 1999

Chris Myers

Example Function: Detection of Signal Events

architecture behavioral of dff is
function rising_edge(signal clock: std_logic) return boolean is
variable edge: boolean := FALSE;
begin
  edge := (clock = '1' and clock'event);
  return edge;
end rising_edge;

begin
  output: process
  begin
    wait until (rising_edge(clk));
    Q <= D after 5 ns;
  ...
Type Conversion Functions

function to_bitvector(svalue:
    std_logic_vector) return bit_vector is
variable outvalue:bit_vector(svalue'length-1
downto 0);
begins
    for i in svalue'range loop
        case svalue(i) is
        when '0' => outvalue(i):='0';
        when '1' => outvalue(i):='1';
        when others => outvalue(i):='0';
        end case;
    end loop;
return outvalue;
end to_bitvector;

Resolution Functions

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
    'I', -- Don't care
);
function resolved(s:std_logic_vector)
    return std_ulogic;
subtype std_logic is resolved std_ulogic;

Example: to_vector Function

Resolution Table

<table>
<thead>
<tr>
<th></th>
<th>'U'</th>
<th>'X'</th>
<th>'0'</th>
<th>'1'</th>
<th>'Z'</th>
<th>'W'</th>
<th>'L'</th>
<th>'H'</th>
<th>'I'</th>
</tr>
</thead>
<tbody>
<tr>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
<td>'U'</td>
</tr>
<tr>
<td>'X'</td>
<td>'U'</td>
<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>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'X'</td>
<td>'0'</td>
<td>'0'</td>
<td>'0'</td>
<td>'0'</td>
<td>'X'</td>
</tr>
<tr>
<td>'1'</td>
<td>'U'</td>
<td>'X'</td>
<td>'1'</td>
<td>'1'</td>
<td>'1'</td>
<td>'1'</td>
<td>'1'</td>
<td>'1'</td>
<td>'X'</td>
</tr>
<tr>
<td>'Z'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'Z'</td>
<td>'W'</td>
<td>'L'</td>
<td>'H'</td>
<td>'X'</td>
</tr>
<tr>
<td>'W'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'W'</td>
<td>'W'</td>
<td>'W'</td>
<td>'W'</td>
<td>'X'</td>
</tr>
<tr>
<td>'L'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'L'</td>
<td>'W'</td>
<td>'L'</td>
<td>'W'</td>
<td>'X'</td>
</tr>
<tr>
<td>'H'</td>
<td>'U'</td>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'H'</td>
<td>'W'</td>
<td>'W'</td>
<td>'H'</td>
<td>'X'</td>
</tr>
<tr>
<td>'I'</td>
<td>'U'</td>
<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>
</tbody>
</table>
Example Procedure: Memory Read

architecture behavioral of CPU is

procedure mread(address: in std_logic_vector(2 downto 0);
signal R: out std_logic;
signal S: in std_logic;
signal ADDR: out std_logic_vector(2 downto 0);
signal DATA: out std_logic_vector(31 downto 0)) is

begin
    ADDR <= address;
    R <= '1';
    wait until S='1';
data <= DO;
    R <= '0';
end mread;

Essentials of Procedures

* Procedures are subprograms that can modify one or more of the input parameters.
* Parameters of mode in can only be read, and parameters of mode out can only be written to.
* Can have parameters of mode inout.
* Types of formals must match actuals.
* Default class for in parameters is constant and for out and inout parameters is variable.
* Variables declared within procedure initialized on entry and do not hold values between calls.

Using Procedures

* Signals cannot be declared within procedures.
* Signals can be passed into procedures.
* Procedures can make assignments to signals not in parameter list (ex. ports in the entity).
  * Should avoid doing this, bad coding style.
* Procedures can also be placed into declarative region of a process.
* Process that calls a procedure with a wait statement cannot have a sensitivity list.

Concurrent and Sequential Procedure Calls

* A concurrent procedure call are those invoked in concurrent area of an architecture.
  * Essentially an encapsulated process.
  * Invoked when there is event on input parameter.
  * Parameters cannot be of variable class.
* A sequential procedure call are those invoked from within a process.
  * Invoked when encountered in sequence in process.
library IEEE;
use IEEE.std_logic_1164.all;
entity serial_adder is
  port(a,b,clk,reset:in std_logic;
       z:out std_logic);
end serial_adder;
architecture structural of serial_adder is
component comb
  port(a,b,c_in:in std_logic;
       z,carry:out std_logic);
end component;
signal s1,s2:std_logic;

begin
  process begin
    dff(clk=>clk,reset=>reset,d=>s2,q=>s1,
         qbar=>open);
    wait on clk, reset, s2;
  end process;
  end structural;

begin
  C1:comb port map(a=>a,b=>b,c_in=>s1,z=>z,
                   carry=>s2);
dff(clk=>clk,reset=>reset,d=>s2,q=>s1,
    qbar=>open);
end structural;

procedure dff(signal d,clk,reset:in std_logic;
              signal q,qbar:out std_logic) is
begin
  if (reset = '0') then
    q <= '0' after 5 ns;
    qbar <= '1' after 5 ns;
  elsif (clk'event and clk='1') then
    q <= 'd' after 5 ns;
    qbar <= (not d) after 5 ns;
  end if;
end dff;

Subprogram and Operator Overloading

* Overloading allows the same subprogram name to be used with different parameters.
dff(clk,d,q,qbar);
dff(clk,d,q,qbar,reset,clear);

* Operators such as "*" and "+" can also be overloaded.
function "*"(arg1,arg2:std_logic_vector)
return std_logic_vector;
function "+"(arg1,arg2:signed)
return signed;
Example: Handshake Procedure

Essentials of Packages

- **Packages** provide for the organization of type definitions, functions, and procedures so they can be shared across distinct VHDL programs.
- **Package declaration** contains type definitions and function and procedure interfaces.
- **Package body** contains the implementations of the functions and procedures.

---

Example: IEEE.std_logic_1164

```vhdl
package std_logic_1164 is
    type std_ulogic is ('U','X','0','1','Z','W','L','H','-');

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

    function resolved(s:std_ulogic_vector) return std_ulogic;

    subtype std_logic is resolved std_ulogic;

    function "and"(l,r:std_logic_vector) return std_logic_vector;

    ...
end std_logic_1164;
```

---

Example: Package Body

```vhdl
package body my_dffs is

    procedure dff(signal d,clk,reset:in std_logic;
                   signal q,qbar:out std_logic) is
    begin
        if (reset = '0') then
            q <= '0' after 5 ns;
            qbar <= '1' after 5 ns;
        elsif (clk'event and clk='1') then
            q <= 'd' after 5 ns;
            qbar <= (not d) after 5 ns;
        end if;
    end dff;
end package;
```
Example: Conversion Package

Essentials of Libraries

* Each design unit—entity, architecture, package body—is placed into a design library.
* Libraries are implemented as directories and are referenced by a logical name.
* To use a library, it must be declared:

```vhd
library IEEE;
use IEEE.std_logic_1164.all -- or .myfunc
```
* STD and WORK are implicitly declared.
* Every design unit must be preceded by library and use clauses (not just at top of file).

Typical Design Description

What We Have Learned So Far

* Functions (type conversion and resolution)
* Procedures (concurrent and sequential calls)
* Subprogram and operator overloading
* Package declarations and bodies
* Libraries
* NEXT TIME: Basic Input/Output