Subprograms

Why have subprograms?
- Design re-use, Sharing, Readability
Subprogram issues:
- How do subprograms execute with respect to simulation time?
  - Wait statements?
  - Signal assignments?
- Concurrent vs. sequential subprograms

Two types of subprograms:
- Functions
- Procedures

VHDL Subprograms

Classes of subprograms
- Function
  - Computes and returns a value
  - Does not modify any arguments
  - Used only in expressions
- Procedure
  - May modify its arguments
  - Does not return a value
  - Sequential or Concurrent Statement

Ports and Subprogram Parameters

Port modes define how signals connect to an entity:
- IN information flows into entity
- OUT information flows out of entity
- INOUT information flows into and out of entity
- BUFFER information flows into and out of entity
- LINKAGE information flows into and out of entity

Subprogram interface parameters are similar to ports

Interface Parameter of mode OUT

OUT parameters can only be assigned:
entity FOO is
  port(D: out Boolean);
end FOO;
architecture BLAH of FOO is
  signal E: Bit;
begin
  D <= 5 > 3;  -- This is correct.
  E <= '1' when D else '0';  -- This is an error.
  end BLAH;
To declare a function in VHDL, specify:
- the name of the function
- the input parameters (if any) (formal parameters)
- the type of the returned value
- any declarations required by the function
- an algorithm for the computation of the returned value

To declare a function in architecture declaration section:
architecture WITHFUNCTION of WHAT is
<declare function(s) here>
begin
<call functions here>
end;

Structure for Function Body Specification

function function_name (function_formal_parameter_list)
return return_type is
function_declaration_part
begin
sequential_statements
return (return_value);
end function_name;

• CONSTRAINT: All parameters in the function formal parameter list must have mode IN.
• No signals allowed in the function declaration part.

Function Example

Type conversion: Bit_vector to Integer:
function BV2INT (BVIN: Bit_vector(7 downto 0))
return Integer is
variable TEMP: Integer := 0;
begin
for i in 0 to 7 loop
if BVIN(i) = '1' then
TEMP := TEMP + 2**i;
end if;
end loop;
return TEMP;
end BV2INT;

Actuals

Another Function Body

function DECODE3TO8 (V: Bit_vector(2 downto 0))
return Bit_vector is
variable RESULT: Bit_vector(7 downto 0) := "00000001";
begin
if v(0) = '1' then RESULT := RESULT(8 downto 0) & '0'; end if;
if v(1) = '1' then RESULT := RESULT(6 downto 0) & "00"; end if;
if v(2) = '1' then RESULT := RESULT(3 downto 0) & "0000"; end if;
return RESULT;
end DECODE3TO8;

Using Functions

architecture FPGA of CONTROL_UNIT is
insert function definition here
signal IAREG, IBREG, ICREG: Bit_vector(9 downto 0);
signal AREGSEL, BREGSEL, CREGSEL: Bit_vector(2 downto 0);
begin
IAREG <= "00" & DECODE3TO8(AREGSEL);
IBREG <= "00" & DECODE3TO8(BREGSEL);
ICREG <= "00" & DECODE3TO8(CREGSEL);
Function Example

architecture BEHAV of DFF is
signal CLK: Bit;
function RISING_EDGE (signal CLOCK: Bit) return Boolean is
variable EDGE: Boolean;
begin
  EDGE := (CLOCK='1' and CLOCK'event);
return( EDGE );
end RISING_EDGE;
begin
\end{verbatim}

Function Example (cont'd)

begin
Fred: process
begin
  wait until (RISING_EDGE(CLK));
  q <= d after 5 ns;
end process Fred;
end behav;
\end{verbatim}

Another example

- Functions can even be used for delays:
  function delay(value: bit; phl: time; plh: time) return time is
  begin
    if value = '1' then
      return plh;
    else
      return phl;
    end if;
  end delay;

- Then elsewhere:
  A <= B and C after delay(B and C, 16 ns, 10 ns);

Function Usage

- Rules for functions:
  - The only allowable mode for parameters is in.
  - The only allowed object classes are constant or signal.
  - If the object class is not specified, it is assumed to be constant.

Procedures

- PROCEDURES are subprograms that can modify one or more of the input parameters.
- Parameters may be of mode IN, OUT or INOUT.
- If the class of a parameter is not explicitly declared, it is assumed that:
  - INS are assumed to be of class CONSTANT
  - OUTs and INOUTs are assumed to be VARIABLE

Procedures (cont'd)

- As with functions, type of formal in declaration must match type of actual when called.
- Variables declared within a procedure are initialized on each procedure call and values do not persist across invocations of the procedure.

Variables declared within procedures are dynamic.
Procedures

To declare a PROCEDURE in VHDL, specify:
- the name of the procedure
- the input and output parameters (if any)
- any declarations required by the procedure itself
- an algorithm

Structure for Procedure Body Specification

procedure procedure_name ( procedure_parameter_list ) is
procedure_declaration_part
begin
sequential_statements
end procedure_name ;

- Parameters in the procedure formal parameter list may have mode IN, OUT, or INOUT.
- No signals allowed in the procedure declaration part.

Example Procedure (1 of 3)

procedure ADSU16P
     (signal A : in Bit_vector(15 downto 0);
      signal B : in Bit_vector(15 downto 0);
      signal ADD : in Bit;
      signal S : out Bit_vector(15 downto 0);
      signal OFL : out Bit);
begin
Formals

Example (2 of 3)

begin
if ADD = '1' then
   CARRY := CIN;
   MB := B;
else
   CARRY := not CIN;
   for i in 0 to 15 loop
      MB(i) := not B(i);
      end loop;
   end if;
end ADSU16P;

Example (3 of 3)

for i in 0 to 15 loop
   MS(i) := MB(i) xor (A(i) xor CARRY);
   CARRY := (MB(i) and A(i)) or (MB(i) and CARRY) or (A(i) and CARRY);
end loop;
compute overflow bit
if (MB(15) = A(15)) then
   OFL := MB(15) xor MS(15);
else
   OFL <= '0';
end if;
S <= MS;
end ADSU16P;

Using Procedures

- Signals cannot be declared in procedures
- Ports are visible within procedures
- Can have WAITs in procedure **
Observations

- Procedures can be called as concurrent statements or as sequential statements within a process.

Procedure Usage

- Rules for procedures:
  - The allowable modes for parameters are in, out, and inout
  - The allowable object classes depend on the mode of the parameter.
  - If the mode is in and if no object class is specified, then constant is assumed
  - If the mode is inout or out and if no object class is specified, then variable is assumed.

Subprogram Overloading

- Same function name can be used to identify a variety of functions or procedures
- Names are the same, but differ in:
  - number of parameter arguments
  - types of individual parameters

Overload Example

- These all have different types:
  - function "and" ( ARG1, ARG2 : Bit ) return Bit;
  - function "and" ( ARG1, ARG2 : Bit_vector ) return Bit_vector;
  - function "and" ( ARG1, ARG2 : Std_logic ) return Std_logic;
  - function "and" ( ARG1, ARG2 : Signed ) return Signed;
  - function "and" ( ARG1, ARG2 : COLOR ) return COLOR;

Overload Example

- Different number of parameters:
  - DFF ( CLK, D, Q );
  - DFF ( CLK, D, Q, QBAR );
  - DFF ( CLK, CLEAR, Q, QBAR );

Overloading Example 2b

- Overloading by type:
  - architecture OLOAD is
    signal A, B, Z : Bit;
    signal AV, BV, ZV : Bit_vector(3 downto 0);
  - begin
    Z <= A and B;
    ZV <= AV and BV;
    ...

More Overloads

- Can also overload other built-in operators:
  - `+`
    ```
    function "+" (A, B : COLOR) return COLOR;
    ```
  - `*`
    ```
    function "*" (A, B : Std_logic_vector) return Std_logic_vector;
    ```

Summary

- Two types of subprograms:
  - Functions
    - Return a single value, cannot modify inputs
    - No side effects
  - Procedures
    - Can return multiple values, modify inputs
    - May have side effects