VHDL Procedure and Function

VLSI CAD
Case Western Reserve University

Dan Saab

SUBPROGRAM

- VHDL provides a subprogram facility to:
  - divide the code into sections
  - Allow repeatedly used code to be referenced multiple times without rewriting
- Two kinds of subprograms:
  - *PROCEDURES*: encapsulates a collection of sequential statements to be executed as a macro.
  - *FUNCTIONS*: encapsulates a collection of sequential statements that compute a result.

Procedures

- Procedures
  - Do not return value
  - Parameter values may be updated
  - Body split into declarative and definition part (similar to process)
  - Unconstrained parameters are possible (array size remains unspecified)
  - Are used as VHDL statements
    - concurrent
    - sequential
- We will examine its:
  - Declaration
  - Invocation

Procedures Declaration

- Used in
  - Package
  - Entity
  - Architecture
  - Process
- Syntax

```
procedure procedure_name
(parameter_list) is
        declarations
begin
        sequential statements
end procedure_name;
```

Example:

```
II: process is
        procedure ReadMemory is
        begin
                        Addressbus <= Memaddress;
                        MemRead <= '1';
                        MemRequest <= '1';
                        wait until MemReady ='1' or Reset = '1';
                        if Reset = '1' then
                                return;
                        end if;
                        MemData := Databus
                        MemRequest <= '0';
                        wait until MemReady = ‘0';
        end procedure ReadMemory;
        begin
                        ... -- initialization
                loop
                        ...
                        ReadMemory;
                        exit when reset = '1';
                        ...
                end loop;
        end process II;
```

Examples

```
procedure DISPLAY_MUX
 (ALARM_TIME, CURRENT_TIME : in digit;
  SHOW_A            : in std_ulogic;
  signal DISPLAY_TIME : out digit) is
begin
  if (SHOW_A = '1') then
    DISPLAY_TIME <= ALARM_TIME;
  else
    DISPLAY_TIME <= CURRENT_TIME;
  end if;
end DISPLAY_MUX;
```

```
procedure sumNoverN is
variable total: real := 0.0;
begin
        assert N'length > 0 severity failure;
        for index in N'range loop
                total := total + N(index);
        end loop;
        average := total / real(N'length);
end procedure sumNoverN;
```

Procedures

- Procedures may be called concurrently or sequentially. A concurrent procedure call executes whenever any of its in or inout parameters change:

```
architecture SUBPROG of DISP_MUX is
  ...
begin
  DISPLAY_MUX (ALARM_TIME, CURRENT_TIME,
               SHOW_A, DISPLAY_TIME);
end SUBPROG;
```

Procedures

- Procedures may have **in, out** or **inout** parameters.
  - These may be signal, variable or constant.
  - Default for
    - **in** parameters is constant.
    - **out** and **inout** are variable.
  - A constant in parameter can be associated with a signal, variable constant or expression when the procedure is called

Example

```
procedure aproc ( vec1, vec2 : in bit vector;result : out boolean ) is
        variable tempi : bit vector(vec1'range) := vec1;
        variable temp2 : bit vector(vec2'range) := vec2;

begin
...
end procedure aproc;
```

```
procedure aproc (a1:in type1;a2:in type2;a3:out type3;a4:in type4) is
begin
...
end procedure aproc;

                aproc(par1, par2, par3, par4 );
                aproc(a1=>par1, a2=>par2, a3=>par3, a4=>par4 );
                aproc(par1, par2, a4=>open, a3=>par3 );
                aproc(par1, par2, par3);
VHDL
```

Procedures

```
procedure negate (a: inout word32 ) is
        variable cin : bit := '1';
        variable cout: bit;
begin
        a := not a;
        for index in a'reverse range loop
                cout := a(index) and cin;
                a(index) := a(index) xor cin;
                cin := cout;
        end loop;
end procedure negate;
```

```
architecture toplevel is
        procedure Npulses (
                width, separation: in delaylength;
                number: in natural := 4 ;
                signal s : out std ulogic ) is
        begin
                for count in 1 to number loop
                                s <= '1', '0' after width;
                                wait for width + separation;
                end loop;
        end procedure Npulses;

begin
        genpulses: process is
        begin
                Npulses (width => period /2,separation => period - period /2,
                                number => pulsecount, s => pulses);
        end process genpulses;
end architecture toplevel;
```

Procedures

- A procedure may declare local variables.
  - These do not retain their values between successive calls
  - are re-initialized each time.

```
procedure PARITY (signal X : in std_ulogic_vector;signal Y : out std_ulogic) is
  variable TMP : std_ulogic := '0';
begin
  for J in X'range loop
    TMP := TMP xor X(J);
  end loop; -- works for any size X
  Y <= TMP;
end PARITY;
```

Example:

```
architecture EXAMPLE of PROCEDURES is
        procedure COUNT ZEROS (A: in bit_vector; signal Q: out integer) is
        variable ZEROS : integer;
        begin
                ZEROS := 0;
                for I in A'range loop
                        if A(I) = '0' then
                                ZEROS := ZEROS +1;
                        end if;
                end loop;
                Q <= ZEROS;
        end COUNT ZEROS;
        signal COUNT: integer;  signal IS_0: boolean;
        begin
                process
                begin
                        IS_0 <= true;
                        COUNT ZEROS("01101001", COUNT);
                        wait for 1 ns;
                        if COUNT > 0 then
                                IS_0 <= false;
                        end if;
                        wait;
                end process;
        end EXAMPLE;
```

Procedure

- If a procedure is defined in a package, its body (the algorithm part) must be placed in the package body.

```
package REF_PACK is
  procedure PARITY
   (signal X : in std_logic_vector;
    signal Y : out std_logic);
end REF_PACK;

package body REF_PACK is
  procedure PARITY
   (signal X : in std_logic_vector;
    signal Y : out std_logic) is
  begin
    -- procedure code
  end PARITY;
end REF_PACK;
```

Procedures call

```
procedure aproc
(a1:in type1;a2:in type2;a3:out type3;a4:in type4)
is
begin
...
end procedure aproc;


aproc(par1, par2, par3, par4 );
aproc(a1=>par1, a2=>par2, a3=>par3, a4=>par4 );
aproc(par1, par2, a4=>open, a3=>par3 );
aproc(par1, par2, par3);
```

```
library IEEE; use IEEE.std logic 1164.all;
entity decoder is port (
        decIn: in std logic vector(1 downto 0);
        decOut: out std logic vector(3 downto 0)
);
end decoder;
architecture simple of decoder is
        procedure DEC2x4 (
                inputs : in std logic vector(1 downto 0);
                decode: out std logic vector(3 downto 0)
                ) is
        begin
                case inputs is
                        when "11" => decode := "1000";
                        when "10" => decode := "0100";
                        when "01" => decode := "0010";
                        when "00" => decode := "0001";
                        when others => decode := "0001";
                end case;
        end DEC2x4;
begin
                DEC2x4(decIn,decOut);
end simple;
```

Functions Syntax

```
function function_name [(parameter_list)]
return type is
        [declarations]
begin
        sequential statements
end function_name [function_name];
```

Functions

- A function in VHDL is a way of defining a new operation that can be used in expressions.
- A function is collection of sequential statements that calculate a result.
  - A function may contain any sequential statement except **signal assignment** and **wait**.
- A function calculates and returns a result that can be used in an expression

Functions

- Declaration used in
  - Package
  - EntityProcess
  - Procedure
- (im)pure declaration optional (default: 'pure')
- Body split into declarative and definition part (similar to process)
- Unconstrained parameters possible (array size remains unspecified)
- Are used as expression in other VHDL statements(concurrent or sequential

procedure call

```
library IEEE; use IEEE.std_logic_1164.all;
entity decoder is port (
    decIn: in std logic vector(1 downto 0);
    decOut: out std logic vector(3 downto 0)
);
end decoder;
architecture simple of decoder is
    procedure DEC2x4 (
        inputs : in std logic vector(1 downto 0);
        decode: out std logic vector(3 downto 0)
        ) is
    begin
        case inputs is
            when "11" => decode := "1000";
            when "10" => decode := "0100";
            when "01" => decode := "0010";
            when "00" => decode := "0001";
            when others => decode := "0001";
        end case;
    end DEC2x4;
begin
    DEC2x4(decIn,decOut); -- concurrent
end simple;                    -- procedure call
```

```
callproc: process is
    DEC2x4(decIn,decOut);
    wait on decIn;
end process callproc;
```

```
library IEEE;
use IEEE.std logic 1164.all;
use IEEE.numeric std.all;
entity powerOfFour is port( clk : in std logic;
        inputVal : in unsigned(3 downto 0);
        power : out unsigned(15 downto 0) );
end powerOfFour;
architecture behavioral of powerOfFour is
        function Pow( N, Exp : integer ) return integer is
        variable Result : integer := 1;
        begin
                for i in 1 to Exp loop
                                Result := Result * N;
                end loop;
        return( Result );
        end Pow;
signal inputValInt: integer range 0 to 15;
signal powerL: integer range 0 to 65535;
begin
        inputValInt <= to_integer(inputVal);
        power <= to_unsigned(powerL,16);
        ...
        process begin
                wait until Clk = '1';
                ...
                powerL <= Pow(inputValInt,4);
                ...
        end process;
end behavioral;
```

Functions

- A function may declare local variables.
  - These do not retain their values between successive calls
  - re-initialized each time it is called.
  - Array-type parameters may be unconstrained:

```
function PARITY (X : std_ulogic_vector)
            return std_ulogic is
  variable TMP : std_ulogic := '0';
begin
        for J in X'range loop
                TMP := TMP xor X(J);
        end loop; -- works for any size X
        return TMP;
end PARITY;
```

Function: Example

```
function bound(a,mim,max:
integer)return integer is
begin
    if a> max then
        return max;
    elsif a < mim then
        return mim;
    else
        return a;
    end if;
end function bound;
```

Functions

- A function can only have input parameters, so the mode (direction) is not specified.

```
function BOOL_TO_SL(X : boolean)
            return std_ulogic is
begin
        if X then
                return '1';
        else
                return '0';
        end if;
end BOOL_TO_SL;
```

Bus Resolution

- VHDL does not allow multiple concurrent signal assignments to the same signal
  - Multiple sequential signal assignments are allowed

```
LIBRARY attlib; USE attlib.att_mvl.ALL;
-- this code will generate an error
ENTITY bus IS
PORT (a, b, c : IN MVL; z : OUT MVL);
END bus;

ARCHITECTURE smoke_generator OF bus IS
SIGNAL circuit_node : MVL;
BEGIN
        circuit_node <= a;
        circuit_node <= b;
        circuit_node <= c;
        z <= circuit_node;
END smoke_generator;
```

Impure Function

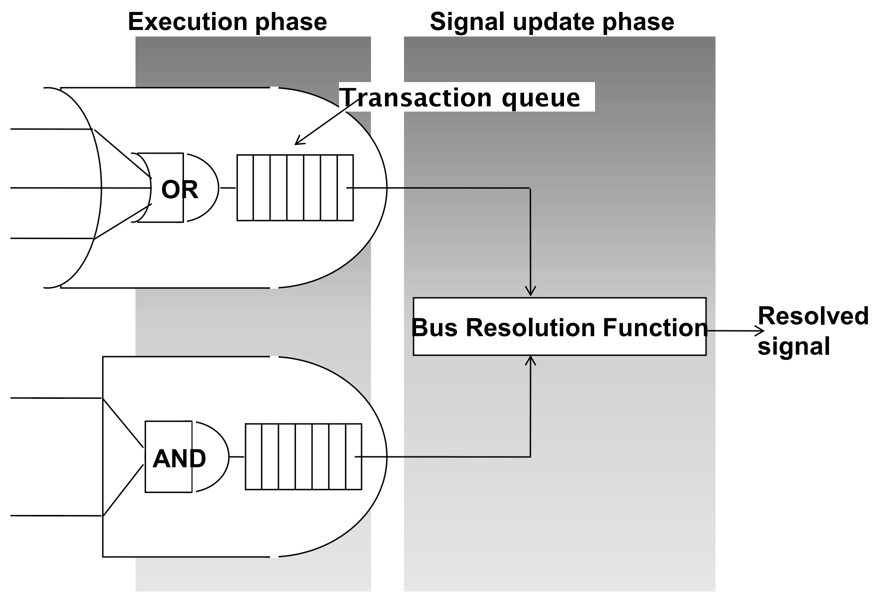
- An impure in the function declaration is a warning to any caller of the function that it might produce different results on different calls, even when passed the same actual parameter values.

```
Aprocess: process is
        constant Nmod : natural := 20;
        subtype Nvalues is natural range 0 to Nmod-1;
        variable Nextv : Nvalues := 1;
        variable a : Nvalues := 1;

        impure function afunc return Nvalues is
                variable N: Nvalues;
        begin
                N := Nextv;
                Nextv := (Nextv + 1) mod Nmod;
                return N;
        end function afunc;

        begin
        ...
        a = a * afunc ;
        ...
end process Aprocess;
```

Signal Resolution and Buses



Functions

- If a function is defined in a package, its body (the algorithm part) must be placed in the package body.

```
package REF_PACK is
        function PARITY (X : bit_vector)
                return bit;
end REF_PACK;

package body REF_PACK is
        function PARITY (X : bit_vector)
                return bit is
        begin
                -- function code
        end PARITY;
end REF_PACK;
```

Blocks and Guards

- Unique to blocks is the GUARD construct
  - A *guarded* signal assignment statement schedules an assignment to the signal driver only if the GUARD expression is true. If the GUARD is false, the corresponding signal drivers are *disconnected*
  - Example

```
ARCHITECTURE guarded_assignments OF n_1_mux IS
                    BEGIN
                     bi: FOR j IN i'RANGE GENERATE
                      bj: BLOCK (s(j)='1' OR s(j)='Z')
                              BEGIN
                                        x <= GUARDED i(j);
                              END BLOCK;
                     END GENERATE;
                    END guarded_assignments
```

Bus Resolution

- A signal which has a bus resolution function associated with it may have multiple drivers

```
LIBRARY attlib; USE attlib.att_mvl.ALL;
USE WORK.bus_resolution.ALL;

ENTITY bus IS
PORT (a, b, c : IN MVL; z : OUT MVL);
END bus;

ARCHITECTURE fixed OF bus IS
SIGNAL circuit_node : wired_and MVL;
BEGIN
     circuit_node <= a;
     circuit_node <= b;
     circuit_node <= c;
     z <= circuit_node;
END fixed;
```

Blocks and Guards

- Blocks are concurrent statements and provide a mechanism to partition an architecture description
- Blocks may be nested to define a hierarchical partitioning of the architectural description
- Items declared in declarative region of block are visible only inside the block, e.g. :
  - signals, subprograms

Bus Resolution Functions

- Are used to determine the assigned value when there are multiple signal drivers to the same signal

```
FUNCTION wired_and (drivers : MVL_VECTOR) RETURN MVL IS
VARIABLE accumulate : MVL := '1';
BEGIN
FOR i IN drivers'RANGE LOOP
     accumulate := accumulate AND drivers(i);
END LOOP;
RETURN accumulate;
END wired_and;
```

- **Bus resolution functions may be user defined or called from a package**

Package

- Collection of dentitions, data-types, subprograms
- Reference made by the design team
- Any changes are known to the team immediately
  - same data types ("downto vs. to")
  - extended functions for all
  - clearing errors for all
- use work.PackageName.all;

Library

- Collection of compiled design

```
library IEEE;
use IEEE.std_logic_1164.all ;
```

- IF an optional guard condition is included, the block becomes a guarded block. the guard condition must return a boolean value, and controls guarded signal assignments within the block.
  - If the guard condition evaluates to false, the drive to any guarded signals from the block is "switched off".
    - Such signals must be declared to be guarded signals of a resolved type.
      - Guarded signals can be declared by adding the words bus or register. The difference between
        - bus requires a resolution function
        - register signal retains its last driven value after all drivers to it have been switched off.

```
architecture BLKS of TRISTATE is
signal INT: std_logic bus;
begin
DRIVER_1: block (EN = '1')
        begin
                INT <= guarded DATA_1;
        end block DRIVER_1;
end BLKS;
```