# DWpackages.vhd

File location:

$VCS_HOME/doc/examples/nativetestbench/systemverilog/vcs_quickstart/DWpackages.vhd

```
--------------------------------------------------------------------------------
--
--       This confidential and proprietary software may be used only
--     as authorized by a licensing agreement from Synopsys Inc.
--     In the event of publication, the following notice is applicable:
--
--                    (C) COPYRIGHT 2003 - 2004 SYNOPSYS INC.
--                           ALL RIGHTS RESERVED
--
--       The entire notice above must be reproduced on all authorized
--     copies.
--
-- AUTHOR:    Derived From Entity Files
--
-- VERSION:   Components package file for DWpackages
--
-- DesignWare_version: 5bfa8d02
-- DesignWare_release: V-2004.06-DWF_0406
--
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_arith.all;

package DWpackages is
  -- Content from file, DWpackages.commonpp
        --------------------------------------------------------
        -- Useful Constants
        --------------------------------------------------------
constant LOW                    : std_logic     := '0';     -- Denotes logic low
        constant HIGH                   : std_logic     := '1';
    -- Denotes logic high

    -- convert an integer to an std_logic_vector
    function DW_CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER) return std_logic_vector;
    function DW_CONV_STD_LOGIC_VECTOR(ARG: UNSIGNED; SIZE: INTEGER) return std_logic_vector;
function DW_CONV_STD_LOGIC_VECTOR(ARG: SIGNED; SIZE: INTEGER) return std_logic_vector;
function DW_CONV_STD_LOGIC_VECTOR(ARG: STD_ULOGIC; SIZE: INTEGER) return std_logic_vecto
    -- This function convert an unsigned/signed signal to an integer.
    -- If the signals have invalid values then a -1;returned.
function DW_CONV_INTEGER(ARG: UNSIGNED) return INTEGER;
function DW_CONV_INTEGER(ARG: SIGNED) return INTEGER;
function "*"(L: UNSIGNED; R: SIGNED) return SIGNED;
function "*"(L: SIGNED; R: UNSIGNED) return SIGNED;
    --------------------------------------------------------
    -- This function takes the integer 'num' as a range of
    -- numbers and returne the min. bits necessary to code
    -- them in binary format.
    -- EXAMPLE:
    --   suppose an array index goes from 0 to 15
    --   that means the range is 16
    --    bit_width(16) = 4;
    -- Note: for num between 9 to 16 the return value is 4
    --       for num = 8, will return 3 !!
    --------------------------------------------------------
     FUNCTION bit_width (num : integer) RETURN integer;
    --------------------------------------------------------
    -- This function takes the integer 'num' and returns the
    -- min width of the binary (base2) number representing it
    -- Example: if there are total_num elements in a memory
    --   block, then the width of the address for this
    --   will be:   bit_width_1(total_num - 1)  <== NOTE minus one
    --
    -- Note: for num between 8 to 15 will return 4, for 16
    --       will return 5 !!
    --------------------------------------------------------
     FUNCTION bit_width_1 (num : integer) RETURN integer;
    --------------------------------------------------------
    -- This function will calculate a the ratio for two
    -- numbers if the two numbers are not interger multiples
    -- of each other, the value returned will be
    -- integer_ratio+1.
            -- Example: If L=9 and R=8 then the return value = 2
    --
            -- This function can be useful for sizing a port that
            -- depends on the size of two other ports.  For example
            -- if a paramterized register of 9 bits is to accessed
            -- on a paramaterized 16 bit bus, this function can be
```

```
                -- used to determine the number of chip selects at
                -- elaboration time.
        --------------------------------------------------------
        FUNCTION pratio (L,R : integer) RETURN integer;
        -- This function determines which bit is set in a one-hot encoded
        -- bus and returns the correspondine integer bit position index
         FUNCTION  hot_bit (hot_encoded_bus : std_logic_vector) RETURN integer;
        -- This function convert an unsigned signal to an integer.
        -- If the signals have invalid values then a -1 is returned.
        FUNCTION conv_integer_x(ARG: UNSIGNED) return INTEGER;
        -- This procedure reports the warning "INVALID_INPUT:
        --    An 'U'|'X'|'W'|'Z'|'-' was detected on (signal name
        -- passed through).
        -- This function returns the INTEGER 0 if all bits of
        -- ARG have known values ('1', '0', 'H', or 'L')
        -- otherwise the INTEGER, 1 is returned.
        --------------------------------------------------------
        FUNCTION any_unknown(ARG: std_logic_vector) RETURN integer;
        --------------------------------------------------------
    -- convert an integer to an std_logic_vector
    function DW_CONV_STD_ULOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER) return STD_ULOGIC_VECTOR
    function DW_CONV_STD_ULOGIC_VECTOR(ARG: UNSIGNED; SIZE: INTEGER) return STD_ULOGIC_VECTO
    function DW_CONV_STD_ULOGIC_VECTOR(ARG: SIGNED; SIZE: INTEGER) return STD_ULOGIC_VECTOR;
    function DW_CONV_STD_ULOGIC_VECTOR(ARG: STD_ULOGIC; SIZE: INTEGER) return STD_ULOGIC_VEC
        -- IF any bit of the bus is not a 0, 1 or U, this function will
        -- assigns output bus to Xs, else the output equals the input.
    FUNCTION bus_U01X ( ARG : std_logic_vector; SIZE : integer ) RETURN std_logic_vector;
--------------------------------------------------------------------------------
--             Function to convert std_logic to only one of '0', '1' or 'X'
--------------------------------------------------------------------------------
 FUNCTION To_01X ( inp1 : std_logic ) RETURN std_logic;
--------------------------------------------------------------------------------
--    Function returns the greater of two integers
--------------------------------------------------------------------------------
    function maximum(L, R: INTEGER) return INTEGER;
--------------------------------------------------------------------------------
--    Function returns the minimum of two integers
--------------------------------------------------------------------------------
    function minimum(L, R: INTEGER) return INTEGER;

end DWpackages;

package body DWpackages is

  -- Content from file, DWpackages.commonpp
--  DWF needs following functions
    -- synopsys synthesis_off
    type tbl_type is array (STD_ULOGIC) of STD_ULOGIC;
    constant tbl_BINARY : tbl_type :=
        ('X', 'X', '0', '1', 'X', 'X', '0', '1', 'X');
    -- synopsys synthesis_on
    function MAKE_BINARY(A : STD_ULOGIC) return STD_ULOGIC is
        -- synopsys built_in SYN_FEED_THRU
    begin
        -- synopsys synthesis_off
            if (IS_X(A)) then
                assert false
report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es)
                severity warning;
                return ('X');
            end if;
            return tbl_BINARY(A);
        -- synopsys synthesis_on
    end;
    function MAKE_BINARY(A : UNSIGNED) return UNSIGNED is
        -- synopsys built_in SYN_FEED_THRU
        variable one_bit : STD_ULOGIC;
        variable result : UNSIGNED (A'range);
    begin
        -- synopsys synthesis_off
            for i in A'range loop
                if (IS_X(A(i))) then
                    assert false
    report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'
                    severity warning;
                    result := (others => 'X');
                    return result;
                end if;
                result(i) := tbl_BINARY(A(i));
            end loop;
            return result;
        -- synopsys synthesis_on
    end;
    function MAKE_BINARY(A : SIGNED) return SIGNED is
        -- synopsys built_in SYN_FEED_THRU
        variable one_bit : STD_ULOGIC;
        variable result : SIGNED (A'range);
```

```
begin
    -- synopsys synthesis_off
        for i in A'range loop
            if (IS_X(A(i))) then
                assert false
report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'
                severity warning;
                result := (others => 'X');
                return result;
            end if;
            result(i) := tbl_BINARY(A(i));
        end loop;
        return result;
    -- synopsys synthesis_on
end;
function MAKE_BINARY(A : UNSIGNED) return std_logic_vector is
    -- synopsys built_in SYN_FEED_THRU
    variable one_bit : STD_ULOGIC;
    variable result : std_logic_vector (A'range);
begin
    -- synopsys synthesis_off
        for i in A'range loop
            if (IS_X(A(i))) then
                assert false
report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'
                severity warning;
                result := (others => 'X');
                return result;
            end if;
            result(i) := tbl_BINARY(A(i));
        end loop;
        return result;
    -- synopsys synthesis_on
end;
function MAKE_BINARY(A : SIGNED) return std_logic_vector is
    -- synopsys built_in SYN_FEED_THRU
    variable one_bit : STD_ULOGIC;
    variable result : std_logic_vector (A'range);
begin
    -- synopsys synthesis_off
        for i in A'range loop
            if (IS_X(A(i))) then
```

```
                assert false
report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'
                severity warning;
                result := (others => 'X');
                return result;
            end if;
            result(i) := tbl_BINARY(A(i));
        end loop;
        return result;
    -- synopsys synthesis_on
end;
function MAKE_BINARY(A : UNSIGNED) return SIGNED is
    -- synopsys built_in SYN_FEED_THRU
    variable one_bit : STD_ULOGIC;
    variable result : SIGNED (A'range);
begin
    -- synopsys synthesis_off
        for i in A'range loop
            if (IS_X(A(i))) then
                assert false
report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'
                severity warning;
                result := (others => 'X');
                return result;
            end if;
            result(i) := tbl_BINARY(A(i));
        end loop;
        return result;
    -- synopsys synthesis_on
end;
-- convert an integer to an std_logic_vector
function DW_CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER) return std_logic_vector i
    variable result: std_logic_vector (SIZE-1 downto 0);
    variable temp: integer;
    -- synopsys built_in SYN_INTEGER_TO_SIGNED
    -- synopsys subpgm_id 381
begin
    -- synopsys synthesis_off
    temp := ARG;
    for i in 0 to SIZE-1 loop
        if (temp mod 2) = 1 then
            result(i) := '1';
            else
                result(i) := '0';
            end if;
            if temp > 0 then
                temp := temp / 2;
            elsif (temp > integer'low) then
                temp := (temp - 1) / 2; -- simulate ASR
            else
                temp := temp / 2; -- simulate ASR
            end if;
        end loop;
        return result;
        -- synopsys synthesis_on
    end;

    function DW_CONV_STD_LOGIC_VECTOR(ARG: UNSIGNED; SIZE: INTEGER) return std_logic_vector
        constant msb: INTEGER := minimum(ARG'length, SIZE) - 1;
        subtype rtype is std_logic_vector (SIZE-1 downto 0);
        variable new_bounds : std_logic_vector (ARG'length-1 downto 0);
        variable result: rtype;
        -- synopsys built_in SYN_ZERO_EXTEND
        -- synopsys subpgm_id 382
    begin
        -- synopsys synthesis_off
        new_bounds := MAKE_BINARY(ARG);
        if (new_bounds(0) = 'X') then
            result := rtype'(others => 'X');
            return result;
        end if;
        result := rtype'(others => '0');
        result(msb downto 0) := new_bounds(msb downto 0);
        return result;
        -- synopsys synthesis_on
    end;

    function DW_CONV_STD_LOGIC_VECTOR(ARG: SIGNED; SIZE: INTEGER) return std_logic_vector is
        constant msb: INTEGER := minimum(ARG'length, SIZE) - 1;
        subtype rtype is std_logic_vector (SIZE-1 downto 0);
        variable new_bounds : std_logic_vector (ARG'length-1 downto 0);
        variable result: rtype;
        -- synopsys built_in SYN_SIGN_EXTEND
        -- synopsys subpgm_id 383
    begin
        -- synopsys synthesis_off
    new_bounds := MAKE_BINARY(ARG);
    if (new_bounds(0) = 'X') then
        result := rtype'(others => 'X');
        return result;
    end if;
    result := rtype'(others => new_bounds(new_bounds'left));
    result(msb downto 0) := new_bounds(msb downto 0);
    return result;
    -- synopsys synthesis_on
end;

function DW_CONV_STD_LOGIC_VECTOR(ARG: STD_ULOGIC; SIZE: INTEGER) return std_logic_vecto
    subtype rtype is std_logic_vector (SIZE-1 downto 0);
    variable result: rtype;
    -- synopsys built_in SYN_ZERO_EXTEND
    -- synopsys subpgm_id 384
begin
    -- synopsys synthesis_off
    result := rtype'(others => '0');
    result(0) := MAKE_BINARY(ARG);
    if (result(0) = 'X') then
        result := rtype'(others => 'X');
    end if;
    return result;
    -- synopsys synthesis_on
end;

    -- This function convert an unsigned/signed signal to an integer.
    -- If the signals have invalid values then a -1 is returned.
function DW_CONV_INTEGER(ARG: UNSIGNED) return INTEGER is
    variable result: INTEGER;
    variable tmp: STD_ULOGIC;
    -- synopsys built_in SYN_UNSIGNED_TO_INTEGER
    -- synopsys subpgm_id 366
begin
    -- synopsys synthesis_off
    assert ARG'length <= 31
        report "ARG is too large in DW_CONV_INTEGER"
        severity FAILURE;
    result := 0;
    for i in ARG'range loop
        result := result * 2;
        tmp := tbl_BINARY(ARG(i));
        if tmp = '1' then
```

```
                result := result + 1;
            elsif tmp = 'X' then
                assert false
report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es)
                severity warning;
                assert false
report "DW_CONV_INTEGER: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, and it ha
                severity WARNING;
                return 0;
            end if;
        end loop;
        return result;
        -- synopsys synthesis_on
    end;

    function DW_CONV_INTEGER(ARG: SIGNED) return INTEGER is
        variable result: INTEGER;
        variable tmp: STD_ULOGIC;
        -- synopsys built_in SYN_SIGNED_TO_INTEGER
        -- synopsys subpgm_id 367
    begin
        -- synopsys synthesis_off
        assert ARG'length <= 32
            report "ARG is too large in DW_CONV_INTEGER"
            severity FAILURE;
        result := 0;
        for i in ARG'range loop
            if i /= ARG'left then
                result := result * 2;
                tmp := tbl_BINARY(ARG(i));
                if tmp = '1' then
                    result := result + 1;
                elsif tmp = 'X' then
                    assert false
    report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'
                    severity warning;
                    assert false
    report "DW_CONV_INTEGER: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, and i
                    severity WARNING;
                    return 0;
                end if;
            end if;
        end loop;
    tmp := MAKE_BINARY(ARG(ARG'left));
    if tmp = '1' then
        if ARG'length = 32 then
            result := (result - 2**30) - 2**30;
        else
            result := result - (2 ** (ARG'length-1));
        end if;
    end if;
    return result;
    -- synopsys synthesis_on
end;
function mult(A,B: SIGNED) return SIGNED is
  variable BA: SIGNED((A'length+B'length-1) downto 0);
  variable PA: SIGNED((A'length+B'length-1) downto 0);
  variable AA: SIGNED(A'length downto 0);
  variable neg: STD_ULOGIC;
  constant one : UNSIGNED(1 downto 0) := "01";

  -- pragma map_to_operator MULT_TC_OP
  -- pragma type_function MULT_SIGNED_ARG
  -- pragma return_port_name Z
  begin
    if (A(A'left) = 'X' or B(B'left) = 'X') then
        PA := (others => 'X');
        return(PA);
    end if;
    PA := (others => '0');
    neg := B(B'left) xor A(A'left);
    BA := CONV_SIGNED(('0' & abs(B)),(A'length+B'length));
    AA := '0' & abs(A);
    for i in 0 to A'length-1 loop
      if AA(i) = '1' then
        PA := PA+BA;
      end if;
      BA := SHL(BA,one);
    end loop;
    if (neg= '1') then
      return(-PA);
    else
      return(PA);
    end if;
  end;
    function "*"(L: UNSIGNED; R: SIGNED) return SIGNED is
        -- pragma label_applies_to mult
        -- synopsys subpgm_id 297
    begin
  return       mult(CONV_SIGNED(L, L'length+1),
                          CONV_SIGNED(R, R'length)); -- pragma label mult
    end;
    function "*"(L: SIGNED; R: UNSIGNED) return SIGNED is
        -- pragma label_applies_to mult
        -- synopsys subpgm_id 298
    begin
        return      mult(CONV_SIGNED(L, L'length),
                         CONV_SIGNED(R, R'length+1)); -- pragma label mult
    end;
        --------------------------------------------------------
        --  This function succesively checks the number
        --  against powers of 2 to see where it fits.
        --  This is a sequential search of log_base2_ceilings
        --   a binary search using half of max length of integer
        --   is probably faster    (argument is RANGE)
        --------------------------------------------------------
         FUNCTION bit_width (num : integer) RETURN integer IS
         variable count : integer;
         Begin
           count := 1;
           if (num <= 0) then return 0;
           elsif (num <= 2**10) then
            for i in 1 to 10 loop
             if (2**count >= num)  then
              return i;
             end if;
             count := count + 1;
            end loop;
           elsif (num <= 2**20) then
            for i in 1 to 20 loop
             if (2**count >= num)  then
              return i;
             end if;
             count := count + 1;
            end loop;
           elsif (num <= 2**30) then
            for i in 1 to 30 loop
```

```
     if (2**count >= num)  then
      return i;
     end if;
     count := count + 1;
    end loop;
   else
    for i in 1 to num loop
     if (2**i >= num)  then
      return i;
     end if;
    end loop;
  end if;
 end bit_width;

--------------------------------------------------------
--   similar to above function (BITS FOR REPRESENTING ARG)
--------------------------------------------------------
 FUNCTION bit_width_1 (num : integer) RETURN integer IS
 Begin
  if (num < 0) then return 0;
  elsif (num = 0) then return 1;
  else
   for i in 1 to num loop
     if (2**i > num) then
        return i;
     end if;
   end loop;
  end if;
 end bit_width_1;
--------------------------------------------------------

--------------------------------------------------------
FUNCTION pratio (L,R : integer) RETURN integer IS
begin
        return (L+R-1)/R;
end pratio;
--------------------------------------------------------
 FUNCTION  hot_bit (hot_encoded_bus : std_logic_vector) RETURN integer IS
--------------------------------------------------------
-- This function determines which bit is set in a one-hot encoded
-- bus and returns the correspondine integer bit position index
-- If the input is not one-hot encoded, then "X" will be returned.
```

```
        -- Indices run from 1..N
        variable num_ones : natural;
        variable pos,ind  : natural;
        variable found    : boolean;
        begin
           num_ones  := 0;
           pos       := 0;
           ind       := 0;
           found     := FALSE;
           for index in hot_encoded_bus'low to  hot_encoded_bus'high loop
              if (hot_encoded_bus(index) = HIGH) then
                 found := TRUE;
                 ind := pos;
                 num_ones := num_ones + 1;
              end if;
              pos := pos + 1;
           end loop;
-- synopsys synthesis_off
           assert (num_ones = 1)
              report "Input Argument to hot_bits is not one-hot encoded"
             severity warning;
-- synopsys synthesis_on
           if (num_ones = 1) then
              return(ind);
           else
              return(1);
           end if;
        end hot_bit;
        --------------------------------------------------------
        FUNCTION conv_integer_x(ARG: UNSIGNED) return INTEGER is
        --------------------------------------------------------
        -- This function is a cut and paste of the IEEE function
        -- CONV_INTEGER_X expect instead of returning a zero for
        -- invalid values, this function returns a -1.
        variable result: INTEGER;
        variable tmp: STD_ULOGIC;

        type tbl_type is array (STD_ULOGIC) of STD_ULOGIC;
        constant tbl_BINARY : tbl_type :=
                 ('X', 'X', '0', '1', 'X', 'X', '0', '1', 'X');
        begin
           -- synopsys synthesis_off
```

```
            assert ARG'length <= 31
                 report "ARG is too large in CONV_INTEGER_X"
                 severity FAILURE;
            result := 0;
            for i in ARG'range loop
                 result := result * 2;
                 tmp := tbl_BINARY(ARG(i));
                 if tmp = '1' then
                    result := result + 1;
                 elsif tmp = 'X' then
                    assert false
    report "There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(
                    severity warning;
                         assert false

 report "CONV_INTEGER_X: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, and it has
                         severity WARNING;
                         return -1;
                  end if;
            end loop;
            return result;
             -- synopsys synthesis_on
         end;
         --------------------------------------------------------
         PROCEDURE invalid_input(
                 CONSTANT signal_name : in STRING) is
         --------------------------------------------------------
         -- This procedure accepts a string to include in the warning message
         begin
 -- synopsys synthesis_off
                  assert false
  report "INVALID_INPUT: An 'U'|'X'|'W'|'Z'|'-' was detected on " & signal_name
                  severity WARNING;
 -- synopsys synthesis_on
         end;

         --------------------------------------------------------
         FUNCTION any_unknown(ARG: std_logic_vector) RETURN integer IS
         --------------------------------------------------------
         -- This function returns the INTEGER 0 if all bits of
         -- ARG have known values ('1', '0', 'H', or 'L')
         -- otherwise the INTEGER, 1 is returned.
         begin
```

```
      -- synopsys synthesis_off
      for i in ARG'range loop
if ((ARG(i)/='0') AND (ARG(i)/='1') AND (ARG(i)/='L') AND (ARG(i)/='H')) then
           return 1;
        end if;
      end loop;
      return 0;
       -- synopsys synthesis_on
   end;
   FUNCTION any_unknown(ARG: unsigned) RETURN integer IS
   begin
      -- synopsys synthesis_off
      for i in ARG'range loop
if ((ARG(i)/='0') AND (ARG(i)/='1') AND (ARG(i)/='L') AND (ARG(i)/='H')) then
           return 1;
        end if;
      end loop;
      return 0;
       -- synopsys synthesis_on
   end;
   FUNCTION any_unknown(ARG: signed) RETURN integer IS
   begin
      -- synopsys synthesis_off
      for i in ARG'range loop
if ((ARG(i)/='0') AND (ARG(i)/='1') AND (ARG(i)/='L') AND (ARG(i)/='H')) then
           return 1;
        end if;
      end loop;
      return 0;
       -- synopsys synthesis_on
   end;
   FUNCTION any_unknown(ARG: std_ulogic_vector) RETURN integer IS
   begin
      -- synopsys synthesis_off
      for i in ARG'range loop
if ((ARG(i)/='0') AND (ARG(i)/='1') AND (ARG(i)/='L') AND (ARG(i)/='H')) then
           return 1;
        end if;
      end loop;
      return 0;
       -- synopsys synthesis_on
```

```
        end;

    -- convert an integer to an std_logic_vector
    function DW_CONV_STD_ULOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER) return STD_ULOGIC_VECTOR
    begin
        return STD_ULOGIC_VECTOR(DW_CONV_STD_LOGIC_VECTOR(ARG, SIZE));
    end;

    function DW_CONV_STD_ULOGIC_VECTOR(ARG: UNSIGNED; SIZE: INTEGER) return STD_ULOGIC_VECTO
    begin
        return STD_ULOGIC_VECTOR(DW_CONV_STD_LOGIC_VECTOR(ARG, SIZE));
    end;

    function DW_CONV_STD_ULOGIC_VECTOR(ARG: SIGNED; SIZE: INTEGER) return STD_ULOGIC_VECTOR
    begin
        return STD_ULOGIC_VECTOR(DW_CONV_STD_LOGIC_VECTOR(ARG, SIZE));
    end;

    function DW_CONV_STD_ULOGIC_VECTOR(ARG: STD_ULOGIC; SIZE: INTEGER) return STD_ULOGIC_VEC
    begin
        return STD_ULOGIC_VECTOR(DW_CONV_STD_LOGIC_VECTOR(ARG, SIZE));
    end;

    FUNCTION bus_U01X ( ARG : std_logic_vector; SIZE : integer ) RETURN std_logic_vector is
      variable result : std_logic_vector(SIZE-1 downto 0);
    BEGIN
          result := (others => 'X');
          for i in ARG'range loop
            if ((ARG(i)/='0') AND (ARG(i)/='1') AND (ARG(i)/='L') AND
               (ARG(i)/='H') AND (ARG(i)/='U')) then
                result := (others => 'X');
            else
                result := ARG;
            end if;
          end loop;
      return result;
    END bus_U01X;

--------------------------------------------------------------------------------
--             Function to convert std_logic to only one of '0', '1' or 'X'
--------------------------------------------------------------------------------

 FUNCTION To_01X ( inp1 : std_logic ) RETURN std_logic is
 variable result : std_logic;
 BEGIN
    result := 'X';
    case (inp1) is
       when 'H' | '1' => result := '1';
          when 'L' | '0' => result := '0';
          when others    => result := 'X';
        end case;
     return result;
     END To_01X;
    --------------------------------------------------------------------------------
    --    Function returns the greater of two integers
    --------------------------------------------------------------------------------
        function maximum(L, R: INTEGER) return INTEGER is
        begin
            if L > R then
               return L;
            else
               return R;
            end if;
        end maximum;

    --------------------------------------------------------------------------------
    --    Function returns the minimum of two integers
    --------------------------------------------------------------------------------

        function minimum(L, R: INTEGER) return INTEGER is
        begin
            if L < R then
               return L;
            else
               return R;
            end if;
        end minimum;
    end DWpackages;
```