EEL 4712
Midterm 1 – Spring 2015
**VERSION 1**

Name: ______________________________

UFID: ______________________________

Sign here if you want your test to be returned in class, where other students might see your score:

______________________________

**IMPORTANT:**
• Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.
• **As always, the best answer gets the most points.**

## COVER SHEET:

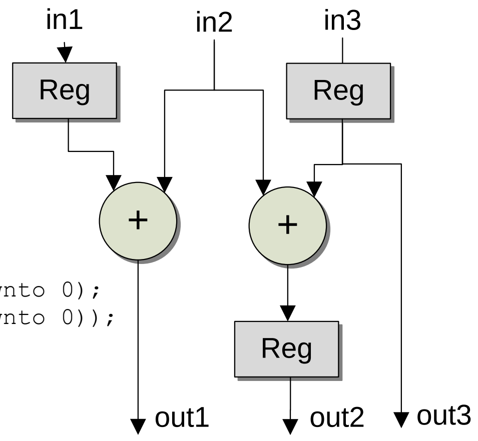
| Problem#: | Points |
|---|---|
| 1 (15 points) | |
| 2 (12 points) | |
| 3 (6 points) | |
| 4 (12 points) | |
| 5 (8 points) | |
| 6 (6 points) | |
| 7 (16 points) | |
| 8 (5 points) | |
| 9 (15 points) | |
| 10 (5 points) | 5 |

**Total:**

**Regrade Info:**

```
ENTITY _entity_name IS
PORT(__input_name, __input_name : IN STD_LOGIC;
__input_vector_name : IN STD_LOGIC_VECTOR(__high downto __low);
__bidir_name, __bidir_name : INOUT STD_LOGIC;
__output_name, __output_name : OUT STD_LOGIC);
END __entity_name;

ARCHITECTURE a OF __entity_name IS
SIGNAL __signal_name : STD_LOGIC;
BEGIN
-- Process Statement
-- Concurrent Signal Assignment
-- Conditional Signal Assignment
-- Selected Signal Assignment
-- Component Instantiation Statement
END a;

__instance_name: __component_name PORT MAP (__component_port => __connect_port,
__component_port => __connect_port);

WITH __expression SELECT
__signal <= __expression WHEN __constant_value,
__expression WHEN __constant_value,
__expression WHEN __constant_value,
__expression WHEN __constant_value;
__signal <= __expression WHEN __boolean_expression ELSE
__expression WHEN __boolean_expression ELSE
__expression;

IF __expression THEN
__statement;
__statement;
ELSIF __expression THEN
__statement;
__statement;
ELSE
__statement;
__statement;
END IF;

CASE __expression IS
WHEN __constant_value =>
__statement;
__statement;
WHEN __constant_value =>
__statement;
__statement;
WHEN OTHERS =>
__statement;
__statement;
END CASE;

<generate_label>: FOR <loop_id> IN <range> GENERATE
-- Concurrent Statement(s)
END GENERATE;

type array_type is array(__upperbound downto __lowerbound);
```

1) (15 points) Fill in the following behavioral VHDL to implement the illustrated circuit. Assume that clk and rst connect to every register. All wires and operations are *width* bits. Ignore overflow from the adders.



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test1 is
    generic (
        width : positive := 8);
    port (
        clk, rst          : in  std_logic;
        in1, in2, in3     : in  std_logic_vector(width-1 downto 0);
        out1, out2, out3 : out std_logic_vector(width-1 downto 0));
end test1;

architecture BHV of test1 is




begin

    process(clk, rst)
    begin
        if (rst = '1') then





        elsif (rising_edge(clk)) then










        end if;
    end process;









end BHV;
```

2) (12 points) Draw the circuit that will be synthesized from the following sequential logic description. You can omit the clk and rst signals. Just show registers and add operations. For partial credit add signal labels to registers.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test2 is
    generic(
        width : positive := 8);
    port (
        clk,rst  : in  std_logic;
        in1, in2, in3  : in  std_logic_vector(width-1 downto 0);
        out1,out2 : out std_logic_vector(width-1 downto 0));
end test2;

architecture BHV of test2 is

    signal regIn1, regIn2, regIn3 : std_logic_vector(width-1 downto 0);
    signal regAddOut1, regAddOut2 : std_logic_vector(width-1 downto 0);
begin
    process(clk, rst)
    begin
        if (rst = '1') then
            regIn1     <= (others => '0');
            regIn2     <= (others => '0');
            regIn3     <= (others => '0');
            regAddOut1 <= (others => '0');
            regAddOut2 <= (others => '0');

        elsif (rising_edge(clk)) then
            regIn1 <= in1;
            regIn2 <= in2;
            regIn3 <= in3;
            regAddOut1 <= std_logic_vector(unsigned(regIn1)+unsigned(regIn2));
            regAddOut2 <= regAddOut1;
        end if;
    end process;

    process(regAddOut1, regAddOut2, regIn3)
    begin
        out1 <= regAddOut1;
        out2 <= std_logic_vector(unsigned(regAddOut2)+unsigned(regIn3));
    end process;
end BHV;
```

3) (6 points) Fill in the CASE_TEST architecture with code that is semantically equivalent to the IF_TEST architecture, but use a case statement instead of an if statement. Hint: use “when others” to include as many conditions as possible.

```
library ieee;
use ieee.std_logic_1164.all;

entity if_case is
    port (
        cond   : in  std_logic_vector(2 downto 0);
        output : out std_logic_vector(1 downto 0));
end if_case;

architecture IF_TEST of if_case is
begin
    process(cond)
    begin
        if (cond(2) = '1') then
            output <= "00";
        elsif (cond(1) = '1') then
            output <= "01";
        elsif (cond(0) = '1') then
            output <= "10";
        else
            output <= "11";
        end if;
    end process;
end IF_TEST;

architecture CASE_TEST of if_case is
begin
    process(cond)
    begin
        case cond is
```

```
        end case;
    end process;
end CASE_TEST;
```

4) a. (6 points) For the IF_TEST architecture in question 3), what synthesis guideline would be violated if you removed the else statement?

b. (6 points) What type of component would synthesis infer if the else statement was removed?

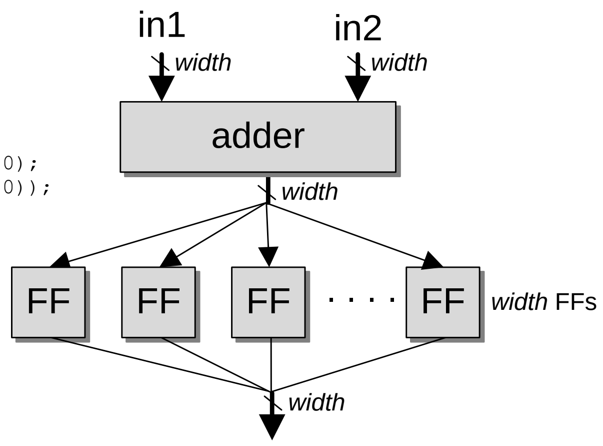
5) (8 points) Identify any violations of the *synthesis coding guidelines for sequential logic*

```
process(clk, rst)
begin
  if (rst = '1') then
    output <= (others => '0');
  elsif (rising_edge(clk)) then
    output <= input;
  end if;

  if (en = '1') then
    output2 <= input;
  end if;
end process;
```

6) (6 points) The following code will generate an error (not a warning) when synthesized in Quartus. Describe the error (hint: it is not a syntax error):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test is
    generic (width : positive := 8);
    port(
        clk        : in  std_logic;
        rst        : in  std_logic;
        in1, in2   : in  std_logic_vector(width-1 downto 0);
        out1, out2 : out std_logic_vector(width-1 downto 0));
end test;

architecture BHV of test is
begin
    process(clk, rst)
    begin
        if (rst = '1') then
            out1 <= (others => '0');
            out2 <= (others => '0');
        elsif (rising_edge(clk)) then
            out1 <= in1;
        end if;
    end process;

    out2 <= std_logic_vector(unsigned(in1)+unsigned(in2));

end BHV;
```

7) (16 points) Fill in the provided code to create the illustrated structural architecture using a series of pre-existing *FF* and *adder* components. Use the component declarations to determine their I/O. Make sure to use the for-generate loop for the *width* FF instances, where each FF connects to a single bit of the adder output and the overall output. Declare any required internal signals.



```
library ieee;
use ieee.std_logic_1164.all;

entity test3 is
    generic(width : positive := 8);
    port (
        clk, rst : in  std_logic;
        in1, in2 : in  std_logic_vector(width-1 downto 0);
        output   : out std_logic_vector(width-1 downto 0));
end test3;

architecture STR of test3 is

    component FF
        port (
            clk, rst, D : in  std_logic;
            Q           : out std_logic);
    end component;

    component adder
        generic(width : positive);
        port (
            in1, in2 : in  std_logic_vector(width-1 downto 0);
            sum      : out std_logic_vector(width-1 downto 0));
    end component;




begin

    U_ADD : adder generic map (width => width)
        port map (









               );

    U_FFS : for i in 0 to width-1 generate

        U_FF : FF port map (










             );




    end generate U_FFS;
end STR;
```

8) (5 points) Why do generics work with a vhd file but not a vho file?

9) a. (9 points) Define the carry out ($c_4$) logic of a 4-bit carry lookahead adder (CLA) in terms of the propagate signals ($p_i$), generate signals ($g_i$), and carry in ($c_0$).

b. (3 points) True/False. Creating a wider CLA by connecting the carry out of one CLA into the carry in of another CLA has a constant propagation delay.

c. (3 points) True/False. Creating a wider CLA by using a tree-based hierarchy of CLAs with carry generation logic at each level (e.g., the hierarchical architecture from lab 3) has a constant propagation delay. List any assumptions.

10) 5 free points for having to take a test at 8:30am.

# 1164 PACKAGES QUICK REFERENCE CARD

Revision 2.1

| | | | |
|---|---|---|---|
| () | Grouping | [ ] | Optional |
| {} | Repeated | \| | Alternative |
| **bold** | As is | CAPS | User Identifier |
| *italic* | VHDL-93 | $_c$ | commutative |

| | | |
|---|---|---|
| b | ::= | BIT |
| bv | ::= | BIT_VECTOR |
| u/l | ::= | STD_ULOGIC/STD_LOGIC |
| uv | ::= | STD_ULOGIC_VECTOR |
| lv | ::= | STD_LOGIC_VECTOR |
| un | ::= | UNSIGNED |
| sg | ::= | SIGNED |
| in | ::= | INTEGER |
| na | ::= | NATURAL |
| sm | ::= | SMALL_INT (subtype INTEGER range 0 to 1) |

## 1. IEEE's STD_LOGIC_1164

### 1.1. Logic Values

| | |
|---|---|
| **'U'** | Uninitialized |
| **'X'/'W'** | Strong/Weak unknown |
| **'0'/'L'** | Strong/Weak 0 |
| **'1'/'H'** | Strong/Weak 1 |
| **'Z'** | High Impedance |
| **'-'** | Don't care |

### 1.2. Predefined Types

| | |
|---|---|
| **STD_ULOGIC** | Base type |
| Subtypes: | |
| **STD_LOGIC** | Resolved STD_ULOGIC |
| **X01** | Resolved X, 0 & 1 |
| **X01Z** | Resolved X, 0, 1 & Z |
| **UX01** | Resolved U, X, 0 & 1 |
| **UX01Z** | Resolved U, X, 0, 1 & Z |

**STD_ULOGIC_VECTOR(**na **to** | **downto** na**)**
Array of STD_ULOGIC
**STD_LOGIC_VECTOR(**na **to** | **downto** na)
Array of STD_LOGIC

### 1.3. Overloaded Operators

| Description | Left | Operator | Right |
|---|---|---|---|
| bitwise-and | u/l,uv,lv | **and**, **nand** | u/l,uv,lv |
| bitwise-or | u/l,uv,lv | **or**, **nor** | u/l,uv,lv |
| bitwise-xor | u/l,uv,lv | **xor**, ***xnor*** | u/l,uv,lv |
| bitwise-not | | **not** | u/l,uv,lv |

### 1.4. Conversion Functions

| From | To | Function |
|---|---|---|
| u/l | b | **TO_BIT(**from[, xmap]**)** |
| uv,lv | bv | **TO_BITVECTOR(**from[, xmap]**)** |
| b | u/l | **TO_STDULOGIC(**from**)** |
| bv,uv | lv | **TO_STDLOGICVECTOR(**from**)** |
| bv,lv | uv | **TO_STDULOGICVECTOR(**from**)** |

## 2. IEEE's NUMERIC_STD

### 2.1. Predefined Types

| | |
|---|---|
| **UNSIGNED(**na **to** \| **downto** na**)** | Array of STD_LOGIC |
| **SIGNED(**na **to** \| **downto** na) | Array of STD_LOGIC |

### 2.2. Overloaded Operators

| Left | Op | Right | Return |
|---|---|---|---|
| | **abs** | sg | sg |
| | **-** | sg | sg |
| un | **+,-,*,/,rem,mod** | un | un |
| sg | **+,-,*,/,rem,mod** | sg | sg |
| un | **+,-,*,/,rem,mod** $_c$ | na | un |
| sg | **+,-,*,/,rem,mod** $_c$ | in | sg |
| un | **<,>,<=,>=,=,/=** | un | bool |
| sg | **<,>,<=,>=,=,/=** | sg | bool |
| un | **<,>,<=,>=,=,/=** $_c$ | na | bool |
| sg | **<,>,<=,>=,=,/=** $_c$ | In | bool |

### 2.3. Predefined Functions

| | |
|---|---|
| **SHIFT_LEFT(**un, na**)** | un |
| **SHIFT_RIGHT(**un, na**)** | un |
| **SHIFT_LEFT(**sg, na**)** | sg |
| **SHIFT_RIGHT(**sg, na**)** | sg |
| **ROTATE_LEFT(**un, na**)** | un |
| **ROTATE_RIGHT(**un, na**)** | un |
| **ROTATE_LEFT(**sg, na**)** | sg |
| **ROTATE_RIGHT(**sg, na**)** | sg |
| **RESIZE(**sg, na**)** | sg |
| **RESIZE(**un, na**)** | un |
| **STD_MATCH(**u/l, u/l**)** | bool |
| **STD_MATCH(**ul, ul**)** | bool |
| **STD_MATCH(**lv, lv**)** | bool |
| **STD_MATCH(**un, un**)** | bool |
| **STD_MATCH(**sg, sg**)** | bool |

### 2.4. Conversion Functions

| From | To | Function |
|---|---|---|
| un,lv | sg | **SIGNED(**from**)** |
| sg,lv | un | **UNSIGNED(**from**)** |
| un,sg | lv | **STD_LOGIC_VECTOR(**from**)** |
| un,sg | in | **TO_INTEGER(**from**)** |
| na | un | **TO_UNSIGNED(**from, size**)** |
| in | sg | **TO_SIGNED(**from, size**)** |

## 3. IEEE's NUMERIC_BIT

### 3.1. Predefined Types

| | |
|---|---|
| **UNSIGNED(**na **to** \| **downto** na**)** | Array of BIT |
| **SIGNED(**na **to** \| **downto** na**)** | Array of BIT |

### 3.2. Overloaded Operators

| Left | Op | Right | Return |
|---|---|---|---|
| | **abs** | sg | sg |
| | **-** | sg | sg |
| un | **+,-,*,/,rem,mod** | un | un |
| sg | **+,-,*,/,rem,mod** | sg | sg |
| un | **+,-,*,/,rem,mod** $_c$ | na | un |
| sg | **+,-,*,/,rem,mod** $_c$ | in | sg |
| un | **<,>,<=,>=,=,/=** | un | bool |
| sg | **<,>,<=,>=,=,/=** | sg | bool |
| un | **<,>,<=,>=,=,/=** $_c$ | na | bool |
| sg | **<,>,<=,>=,=,/=** $_c$ | in | bool |

### 3.3. Predefined Functions

| | |
|---|---|
| **SHIFT_LEFT(**un, na**)** | un |
| **SHIFT_RIGHT(**un, na**)** | un |
| **SHIFT_LEFT(**sg, na**)** | sg |
| **SHIFT_RIGHT(**sg, na**)** | sg |
| **ROTATE_LEFT(**un, na**)** | un |
| **ROTATE_RIGHT(**un, na**)** | un |
| **ROTATE_LEFT(**sg, na**)** | sg |
| **ROTATE_RIGHT(**sg, na**)** | sg |
| **RESIZE(**sg, na**)** | sg |
| **RESIZE(**un, na**)** | un |

### 3.4. Conversion Functions

| From | To | Function |
|---|---|---|
| un,bv | sg | **SIGNED(**from**)** |
| sg,bv | un | **UNSIGNED(**from**)** |
| un,sg | bv | **BIT_VECTOR(**from**)** |
| un,sg | in | **TO_INTEGER(**from**)** |
| na | un | **TO_UNSIGNED(**from**)** |
| in | sg | **TO_SIGNED(**from**)** |

## 4. Synopsys' STD_LOGIC_ARITH

### 4.1. Predefined Types

| | |
|---|---|
| **UNSIGNED(**na **to** \| **downto** na**)** | Array of STD_LOGIC |
| **SIGNED(**na **to** \| **downto** na**)** | Array of STD_LOGIC |
| **SMALL_INT** | Integer subtype, 0 or 1 |

### 4.2. Overloaded Operators

| Left | Op | Right | Return |
|---|---|---|---|
| | **abs** | sg | sg,lv |
| | **-** | sg | sg,lv |
| un | **+,-,*,/** | un | un,lv |
| sg | **+,-,*,/** | sg | sg,lv |
| sg | **+,-,*,/** $_c$ | un | sg,lv |
| un | **+,-** $_c$ | in | un,lv |
| sg | **+,-** $_c$ | in | sg,lv |
| un | **+,-** $_c$ | u/l | un,lv |
| sg | **+,-** $_c$ | u/l | sg,lv |
| un | **<,>,<=,>=,=,/=** | un | bool |
| sg | **<,>,<=,>=,=,/=** | sg | bool |
| un | **<,>,<=,>=,=,/=** $_c$ | in | bool |
| sg | **<,>,<=,>=,=,/=** $_c$ | in | bool |

### 4.3. Predefined Functions

| | | | |
|---|---|---|---|
| **SHL(**un, un**)** | un | **SHR(**un, un**)** | un |
| **SHL(**sg, un**)** | sg | **SHR(**sg, un**)** | sg |
| **EXT(**lv, in**)** | lv | zero-extend | |
| **SEXT(**lv, in**)** | lv | sign-extend | |

### 4.4. Conversion Functions

| From | To | Function |
|---|---|---|
| un,lv | sg | **SIGNED(**from**)** |
| sg,lv | un | **UNSIGNED(**from**)** |
| sg,un | lv | **STD_LOGIC_VECTOR(**from**)** |
| un,sg | in | **CONV_INTEGER(**from**)** |
| in,un,sg,u | un | **CONV_UNSIGNED(**from, size**)** |
| in,un,sg,u | sg | **CONV_SIGNED(**from, size**)** |
| in,un,sg,u | lv | **CONV_STD_LOGIC_VECTOR(**from, size**)** |

## 5. Synopsys' STD_LOGIC_UNSIGNED

### 5.1. Overloaded Operators

| Left | Op | Right | Return |
|---|---|---|---|
| | **+** | lv | lv |
| lv | **+,-,*** | lv | lv |
| lv | **+,-** $_c$ | in | lv |
| lv | **+,-** $_c$ | u/l | lv |
| lv | **<,>,<=,>=,=,/=** | lv | bool |
| lv | **<,>,<=,>=,=,/=** $_c$ | in | bool |

### 5.2. Conversion Functions

| From | To | Function |
|---|---|---|
| lv | in | **CONV_INTEGER(**from**)** |

## 6. Synopsys' STD_LOGIC_SIGNED

### 6.1. Overloaded Operators

| Left | Op | Right | Return |
|---|---|---|---|
| | **abs** | lv | lv |
| | **+,-** | lv | lv |
| lv | **+,-,*** | lv | lv |
| lv | **+,-** $_c$ | in | lv |
| lv | **+,-** $_c$ | u/l | lv |
| lv | **<,>,<=,>=,=,/=** | lv | bool |
| lv | **<,>,<=,>=,=,/=** $_c$ | in | bool |

### 6.2. Conversion Functions

| From | To | Function |
|---|---|---|
| lv | in | **CONV_INTEGER(**from**)** |

## 7. Synopsys' STD_LOGIC_MISC

### 7.1. Predefined Functions

| | |
|---|---|
| **AND_REDUCE(**lv \| uv**)** | u/l |
| **OR_REDUCE(**lv \| uv**)** | u/l |
| **XOR_REDUCE(**lv \| uv**)** | u/l |

## 8. Cadence's STD_LOGIC_ARITH

### 8.1. Overloaded Operators

| Left | Op | Right | Return |
|---|---|---|---|
| u/l | **+,-,*,/** | u/l | u/l |
| lv | **+,-,*,/** | lv | lv |
| lv | **+,-,*,/** $_c$ | u/l | lv |
| lv | **+,-** $_c$ | in | lv |
| uv | **+,-,*** | uv | uv |
| uv | **+,-,*** $_c$ | u/l | uv |
| uv | **+,-** $_c$ | in | uv |
| lv | **<,>,<=,>=,=,/=** $_c$ | in | bool |
| uv | **<,>,<=,>=,=,/=** $_c$ | in | bool |

### 8.2. Predefined Functions

| | |
|---|---|
| **SH_LEFT(**lv, na**)** | lv |
| **SH_LEFT(**uv, na**)** | uv |
| **SH_RIGHT(**lv, na**)** | lv |
| **SH_RIGHT(**uv, na**)** | uv |
| **ALIGN_SIZE(**lv, na**)** | lv |
| **ALIGN_SIZE(**uv, na**)** | uv |
| **ALIGN_SIZE(**u/l, na**)** | lv,uv |

C-like ?: replacements:

| | |
|---|---|
| **COND_OP(**bool, lv, lv**)** | lv |
| **COND_OP(**bool, uv, uv**)** | uv |
| **COND(**bool, u/l, u/l**)** | u/l |

### 8.3. Conversion Functions

| From | To | Function |
|---|---|---|
| lv,uv,u/l | in | **TO_INTEGER(**from**)** |
| in | lv | **TO_STDLOGICVECTOR(**from, size**)** |
| in | uv | **TO_STDULOGICVECTOR(**from, size**)** |

## 9. Mentor's STD_LOGIC_ARITH

### 9.1. Predefined Types

| | |
|---|---|
| **UNSIGNED(**na **to** \| **downto** na**)** | Array of STD_LOGIC |
| **SIGNED(**na **to** \| **downto** na**)** | Array of STD_LOGIC |

### 9.2. Overloaded Operators

| Left | Op | Right | Return |
|---|---|---|---|
| | **abs** | sg | sg |
| | **-** | sg | sg |
| u/l | **+,-** | u/l | u/l |
| uv | **+,-,*,/,mod,rem,**** | uv | uv |
| lv | **+,-,*,/,mod,rem,**** | lv | lv |
| un | **+,-,*,/,mod,rem,**** | un | un |
| sg | **+,-,*,/,mod,rem,**** | sg | sg |
| un | **<,>,<=,>=,=,/=** | un | bool |
| sg | **<,>,<=,>=,=,/=** | sg | bool |
| | **not** | un | un |
| | **not** | sg | sg |
| un | **and**,**nand**,**or**,**nor**,**xor** | un | un |
| sg | **and**,**nand**,**or**,**nor**,**xor**,***xnor*** | sg | sg |
| uv | ***sla***,***sra***,***sll***,***srl***,***rol***,***ror*** | uv | uv |
| lv | ***sla***,***sra***,***sll***,***srl***,***rol***,***ror*** | lv | lv |
| un | ***sla***,***sra***,***sll***,***srl***,***rol***,***ror*** | un | un |
| sg | ***sla***,***sra***,***sll***,***srl***,***rol***,***ror*** | sg | sg |

### 9.3. Predefined Functions

| | |
|---|---|
| **ZERO_EXTEND(**uv \| lv \| un, na**)** | same |
| **ZERO_EXTEND(**u/l, na**)** | lv |
| **SIGN_EXTEND(**sg, na**)** | sg |
| **AND_REDUCE(**uv \| lv \| un \| sg**)** | u/l |
| **OR_REDUCE(**uv \| lv \| un \| sg**)** | u/l |
| **XOR_REDUCE(**uv \| lv \| un \| sg**)** | u/l |

### 9.4. Conversion Functions

| From | To | Function |
|---|---|---|
| u/l,uv,lv,un,sg | in | **TO_INTEGER(**from**)** |
| u/l,uv,lv,un,sg | in | **CONV_INTEGER(**from**)** |
| bool | u/l | **TO_STDLOGIC(**from**)** |
| na | un | **TO_UNSIGNED(**from,size**)** |
| na | un | **CONV_UNSIGNED(**from,size**)** |
| in | sg | **TO_SIGNED(**from,size**)** |
| in | sg | **CONV_SIGNED(**from,size**)** |
| na | lv | **TO_STDLOGICVECTOR(**from,size**)** |
| na | uv | **TO_STDULOGICVECTOR(**from,size**)** |