EEL 4712
Midterm 1 – Spring 2010
VERSION 1

Name: _____

UFID: _____

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 (8 points)	
2 (8 points)	
3 (16 points)	
4 (16 points)	
5 (16 points)	
6 (16 points)	
7 (16 points)	
8 (8 points)	

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;
```

1) (8 points) For the entity given below, explain how the generic WIDTH gets its value:

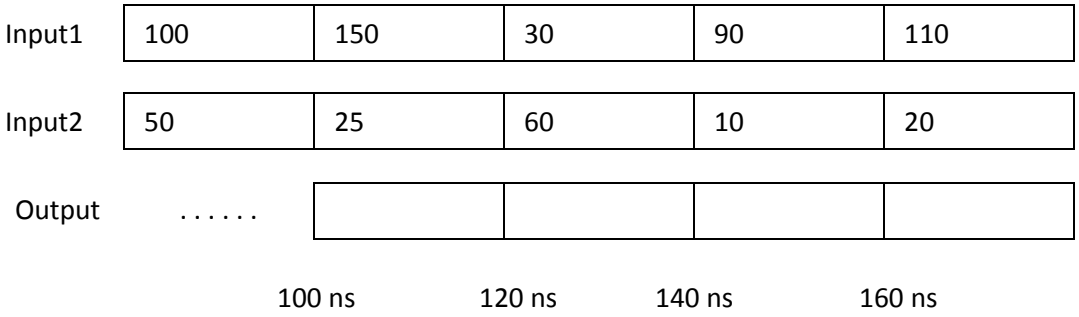
```
library ieee;
use ieee.std_logic_1164.all;

entity ALU is
  generic (
    WIDTH          :     positive := 16);
  port (
    input1, input2 : in  std_logic_vector(WIDTH-1 downto 0);
    sel            : in  std_logic_vector(3 downto 0);
    output         : out std_logic_vector(WIDTH-1 downto 0));
end ALU;
```

2) (8 points) Describe the violation of the synthesis guidelines for *combinational logic* in the following example:

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

entity MUX is
  port (
    input1, input2 : in  std_logic_vector(15 downto 0);
    sel            : in  std_logic;
    en             : in  std_logic;
    output         : out std_logic_vector(15 downto 0));
end MUX;

architecture BHV of MUX is
begin  -- BHV

  process(input1, input2, sel)
  begin
    if en = '1' then
      output <= (others => '0');
    elsif sel = '0' then
      output <= input1;
    else sel = '1' then
      output <= input2;
    end if;
  end process;

end BHV;
```

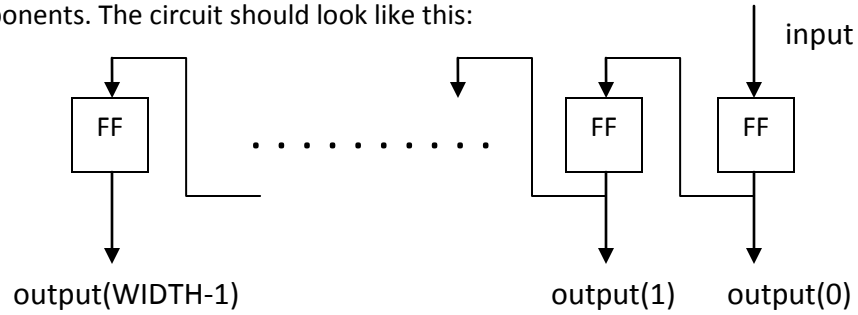
3) A. (8 points) For the following code, fill in the waveform for "output" assuming the values shown are in decimal format:

```
entity ADD is
  port (
    input1, input2 : in  std_logic_vector(15 downto 0);
    output         : out std_logic_vector(15 downto 0);
    overflow       : out std_logic);
end ADD;

architecture BHV of ADD is

  signal temp : unsigned(16 downto 0);
begin
  process(input1, input2)
  begin
    temp     <= unsigned("0"&input1) + unsigned("0"&input2);
    output   <= std_logic_vector(temp(15 downto 0));
    overflow <= std_logic(temp(16));
  end process;
end BHV;
```



B. (8 points) Briefly explain the reason for the behavior of "output", specifically mentioning when a signal's value gets updated.

4) (16 points) Fill in the code provided below to create a shift register with generic width. *You must use a structural architecture with the provided generate loop* that connects together the flip-flop (FF) components. The circuit should look like this:

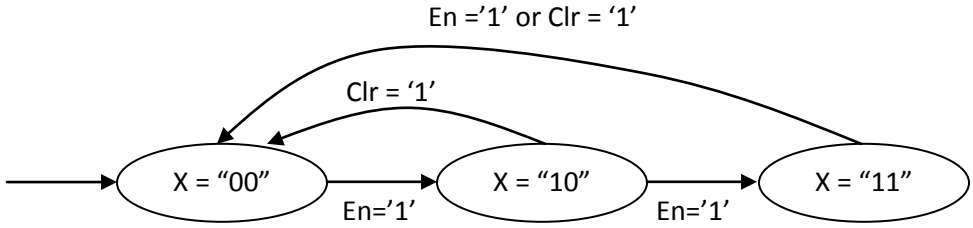



```
library ieee;
use ieee.std_logic_1164.all;

entity SH_REG is
  generic (WIDTH    :     positive := 16);
  port (
    clk, rst : in  std_logic;
    input    : in  std_logic;
    output   : out std_logic_vector(WIDTH-1 downto 0));
end SH_REG;

architecture STR of SH_REG is

  component FF
    port (
      clk, rst : in  std_logic;
      input    : in  std_logic;
      output   : out std_logic);
  end component;



begin  -- STR



  U_SH_REG : for i in 0 to WIDTH-1 generate

    U_FF : FF port map (
      clk    => clk,
      rst    => rst,









      );
  end generate U_SH_REG;



end STR;
```

5) (16 points) Draw a schematic for the register-transfer-level (RTL) circuit that will be synthesized from the following VHDL code. Clearly label all components, inputs, and outputs.

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

entity BHV_TEST is
  port (
    clk, rst       : in  std_logic;
    input1, input2 : in  std_logic_vector(15 downto 0);
    input3, input4 : in  std_logic_vector(15 downto 0);
    output         : out std_logic_vector(15 downto 0));
end BHV_TEST;

architecture BHV of BHV_TEST is

  signal add1, add2 : std_logic_vector(15 downto 0);
  signal mult1 : std_logic_vector(31 downto 0);
begin
  process(clk, rst)
  begin
    if (rst = '1') then
      add1   <= (others => '0');
      add2   <= (others => '0');
      mult1  <= (others => '0');

    elsif (clk'event and clk = '1') then
      add1  <= std_logic_vector(unsigned(input1)+unsigned(input2));
      add2  <= add1;
      mult1 <= std_logic_vector(unsigned(input3)*unsigned(input4));

    end if;
  end process;

  output <= std_logic_vector(unsigned(add2)+
                             unsigned(mult1(15 downto 0)));
end BHV;
```

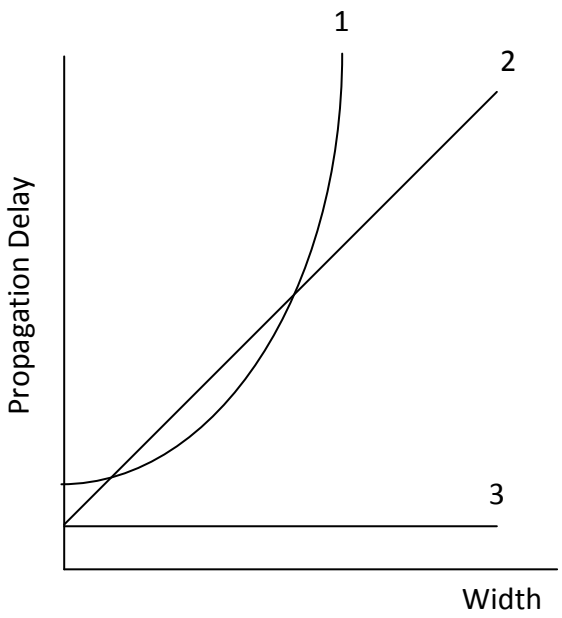
6) (16 points) Fill in the skeleton code to implement the following Moore finite state machine, *using the 1-process FSM model*. Assume the "Clr" has a higher priority than "En", so that if "Clr" is asserted, the "Clr" edges are taken instead of the "En" edges. For other possible conditions that are not shown, assume that each state transitions back to itself (e.g. en='0'). Use the next page if extra room is needed.

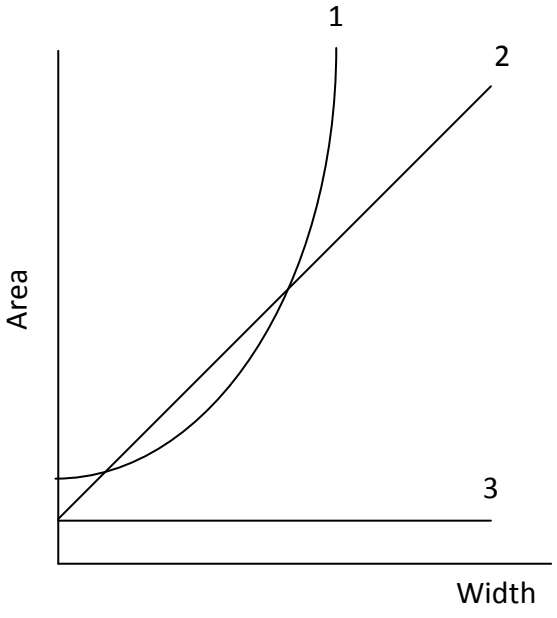



```
library ieee;
use ieee.std_logic_1164.all;

entity FSM is
  port (
    clk, rst, en, clr : in  std_logic;
    x                 : out std_logic_vector(1 downto 0));
end FSM;

architecture BHV of FSM is

  type STATE_TYPE is (


                           );
  signal state : STATE_TYPE;


begin  -- BHV
```

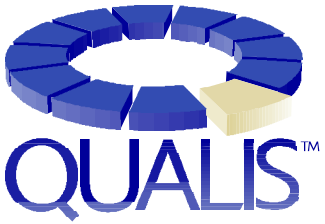
7) a. (8 points) Define each carry bit of a 3-bit carry lookahead adder (i.e., c(1), c(2), c(3)) in terms of the propagate and generate functions, and the carry in (i.e., c(0)).

b. (8 points) For each graph below, circle the number that most closely represents a carry lookahead adder.





8) (8 free points) In the space below, describe every difference between numeric_std and std_logic_arith. Alternatively, leave blank for full credit.



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**)**

Qualis Design Corporation

Elite Consulting and Training in High-Level Design

Phone: +1-503-670-7200 FAX: +1-503-670-0809

E-mail: info@qualis.com Web: http://www.qualis.com