VHDL Arithmetic Functions and IEEE Numeric_std package

Smith ASIC text sections: 12.6.5, 12.6.9, 12.6.10
Synthesizing arithmetic circuits
(12.6.5, 12.6.9, 12.6.10)

- Synthesis tool recognizes overloaded operators and generates corresponding circuits: 
  "+", "-", "*", and "abs" (add, sub, multiply, abs. value)

- Special operations:
  "+1", "-1", unary "+-" (increment, decrement, negate)

- Relational Operators:
  "=", "=/="", "<"", ">"", "<="", ">="

- Use ranged integers instead of unbound to minimize generated logic.

  ```
  signal i : integer range 0 to 15; --4-bit number
  ```
Leonardo restrictions

- Non-integer data types (std_logic_vector) require operator overloading to produce arithmetic circuits
  - A function must be supplied to produce a result for operands having designated data types
  - IEEE library packages define arithmetic functions for certain combinations of data types
- Multiply operator “*” will produce a multiplier, but more efficient technology-specific modules may be better.
- Divide operator “/” only works if dividing by a power of 2, unless using a technology-specific module
Behavioral model of an adder

entity adder is
  port ( a: in integer; -- abstract data type
  b: in integer;
  sum: out integer);
end adder;
architecture behave of adder is
begin
  sum <= a + b; -- synthesis produces adder circuit
end;
Adders/subtracters automatically generated for integer data

variable a,b,c: integer;
c := a + b; -- produces 32-bit adder (signed)

variable a,b,c: integer range 0 to 255;
c := a + b; -- produces 8-bit adder (unsigned)

Constant operands result in reduced logic by removing logic due to hard-wired values.
Ex:   c := a + 5;
Support for arithmetic models (preferred):

- **numeric_std** (ieee library package)
  - defines UNSIGNED and SIGNED as arrays of std_logic
    
    ```
    type SIGNED is array(NATURAL range <>) of STD_LOGIC;
    type UNSIGNED is array(NATURAL range <>) of STD_LOGIC;
    ```
  - defines standard arithmetic/relational operators on these types

Lesser-used packages:

- **numeric_bit**
  - same as above except SIGNED/UNSIGNED are arrays of type bit

- **std_logic_arith** (*from Synopsis*)
  - Non-standard predecessor of numeric_std/numeric_bit
NUMERIC_STD package contents

- Arithmetic functions: + - * / rem mod
  - Combinations of operands for which operators defined:
    - SIGNED + SIGNED return SIGNED
    - SIGNED + INTEGER return SIGNED
    - INTEGER + SIGNED return SIGNED
  - 6 relational operators for above operand combo’s
- Convert between data types:
  - TO_INTEGER(signed), TO_SIGNED(integer,#bits)
  - RESIZE(signed,#bits) – changes # bits
- SHIFT_LEFT/RIGHT, ROTATE_LEFT/RIGHT
- PLUS: all of above with UNSIGNED (NATURAL)
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity Adder4 is
    port ( in1, in2 : in  STD_LOGIC_VECTOR(3 downto 0);
           mySum : out STD_LOGIC_VECTOR(3 downto 0) );
end Adder4;

architecture Behave_B of Adder4 is
begin
    mySum <= STD_LOGIC_VECTOR(SIGNED(in1) + SIGNED(in2));
end Behave_B;

Conversion of closely-related types
For arrays having same dimension and elements of same type
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity Adder4 is
  port ( in1, in2 : in UNSIGNED(3 downto 0) ;
       mySum : out UNSIGNED(3 downto 0) );
end Adder4;

architecture Behave_B of Adder4 is
begin
  mySum <= in1 + in2; -- overloaded '+’ operator
end Behave_B;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
ENTITY counter IS
  port( Q:  out std_logic_vector(3 downto 0);
  ....
ARCHITECTURE behavior OF counter IS
  signal Qinternal:  unsigned(3 downto 0);
begin
  Qinternal <= Qinternal + 1;                -- + from numeric_std
  Q <= std_logic_vector(Qinternal);    -- unsigned->std_logic
  ....
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity Adder_1 is
  port ( A, B : in  UNSIGNED(3 downto 0) ;
         C :     out UNSIGNED(4 downto 0) ) ; -- C(4) = carry
end Adder_1;

architecture Synthesis_1 of Adder_1 is
begin                                    -- A+B could produce a carry
  C <= ('0' & A) + ('0' & B); -- leading ‘0’ balances # bits
end Behave_B;

-- can also use:  C <= resize(A,5) + resize(B,5)
Add/Sub “Accumulator”

-- result_t , xin, addout are UNSIGNED
with addsub select – combinational add/sub
    addout <= (xin + result_t) when '1',
              (xin - result_t) when '0',
              (others => '-') when others;

process (clr, clk) begin -- register part
    if (clr = '0') then
        result_t <= (others => '0');
    elsif rising_edge(clk) then
        result_t <= addout;
    end if;
end process;
-- “when others” creates exhaustive list of choices
Simplified “accumulator” model

-- signal addsub eliminated – circuit will be the same

process (clr, clk) begin
  if (clr = '0') then
    result_t <= (others => '0');
  elsif rising_edge(clk) then
    case addsub is
      when '1' => result_t <= (xin + result_t);
      when '0' => result_t <= (xin - result_t);
      when others => result_t <= (others => '-');
    end case;
  end if;
end process;
Multiple adder structures

\[ z \leq a + b + c + d; \]
--- 3 adders stacked 3 deep

\[ z \leq (a + b) + (c + d); \]
--- 3 adders stacked 2 deep
Resource sharing for mutually-exclusive operations

process (a,b,c,test) begin
  if (test=TRUE) then
    o <= a + b ;   -- either this evaluates
  else
    o <= a + c ;   -- or this evaluates
  end if ;
end process ;
-- Leonardo generates two adders & one mux
Equivalent model

process (a,b,c,test) begin
    variable tmp : integer range 0 to 255 ;
    begin
        if (test=TRUE) then -- mux will select b or c
            tmp := b ;
        else
            tmp := c ;
        end if ;
        o <= a + tmp ;  -- mux output added to a
    end process ;
-- one adder and one mux generated