VHDL Modeling for Synthesis

Arithmetic Functions and IEEE numeric_std package

Text: 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 <u>ranged</u> 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;`

IEEE Std. 1076.3 Synthesis Libraries (12.6.5, 12.6.9, 12.6.10)

- Support for arithmetic models (preferred):
  - ieee.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:
  - ieee.numeric_bit
    - same as above except SIGNED/UNSIGNED are arrays of type bit
  - ieee.std_logic_arith *(from Synopsis)*
    - <u>Non-standard predecessor</u> of numeric_std/numeric_bit

NUMERIC_STD package contents

- Arithmetic functions: + - * / rem mod
  - Combinations of operands for which operators are defined:
    - SIGNED + SIGNED return SIGNED
    - SIGNED + INTEGER return SIGNED
    - INTEGER + SIGNED return SIGNED
    - SIGNED + STD_LOGIC return SIGNED
  - PLUS: above combinations with UNSIGNED and NATURAL
- Other operators for SIGNED/UNSIGNED types:
  - Relational: = /= < > <= >=
  - Shift/rotate: sll, srl, sla, sra, rol, ror
  - Maximum(a,b), Minimum(a,b)
- Convert between types:
  - TO_INTEGER(SIGNED), TO_INTEGER(UNSIGNED)
  - TO_SIGNED(INTEGER,#bits), TO_UNSIGNED(NATURAL,#bits)
  - RESIZE(SIGNED or UNSIGNED,#bits) – changes # bits in the vector

RTL models of arithmetic functions

```
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;
```

Conversion of "closely-related" types

- STD_LOGIC_VECTOR, SIGNED, UNSIGNED:
  - All three types are arrays of STD_LOGIC elements
  - Example: Consider how to interpret "1001"
    - STD_LOGIC_VECTOR: simple array of four bits
    - SIGNED: array of bits representing the number -7 (2's complement #)
    - UNSIGNED: array of bits representing the number 9 (unsigned #)
- Vectors of the same element types may be "converted" (re-typed/re-cast) from one type to another

```
signal A:  std_logic_vector(3 downto 0) := "1001";
signal B:  signed(3 downto 0);
signal C:  unsigned(3 downto 0);
B <= signed(A);               -- interpret A value "1001" as number -7
C <= unsigned(A);             -- interpret A value "1001" as number 9
A <= std_logic_vector(B);     -- interpret B as bit pattern "1001"
```

Conversion of "closely-related" types

For arrays of same dimension, *having elements of same type*

```
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;
```

Interpret STD_LOGIC_VECTOR as SIGNED for function: SIGNED = SIGNED + SIGNED.

Interpret SIGNED result as STD_LOGIC_VECTOR.

Example – binary counter

```
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);
          ….
END counter;

ARCHITECTURE behavior OF counter IS
  signal Qinternal:  unsigned(3 downto 0);
begin
   Qinternal <= Qinternal + 1;            -- "+" defined in numeric_std**
   Q <= std_logic_vector(Qinternal);    -- re-type unsigned as std_logic_vector
  ….
```

** numeric_std defines "+" function: UNSIGNED = UNSIGNED + NATURAL

Using a “variable” to describe sequential behavior within a process

```
-- assume Din and Dout are std_logic_vector
-- and numeric_std package is included
cnt: process(clk)
        variable count: integer;  -- internal counter state
    begin                          -- valid only within a process
        if clk=‘1’ and clk’event then
          if ld=‘1’ then
                count := to_integer(unsigned(Din));  --update immediately
          elsif cnt=‘1’ then
                count := count + 1;                  --update immediately
          end if;
        end if;
        Dout <= std_logic_vector(to_unsigned(count,32)); --schedule ΔDout
   end process;
```

Using a “variable” to describe sequential behavior within a process

```
-- assume Din and Dout are std_logic_vector
-- and numeric_std package is included
cnt: process(clk)
        variable count: integer;  -- internal counter state
    begin                          -- valid only within a process
        if clk=‘1’ and clk’event then
          if ld=‘1’ then
                count := to_integer(unsigned(Din));  --update immediately
          elsif cnt=‘1’ then
                count := count + 1;                  --update immediately
          end if;
        end if;
        Dout <= std_logic_vector(to_unsigned(count,32)); --schedule ΔDout
   end process;
```

Add/Sub "Accumulator"

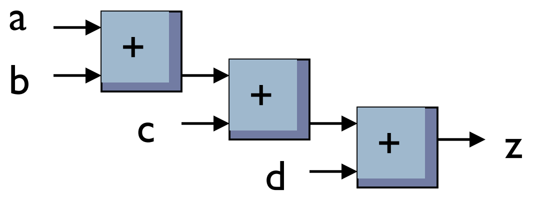
```
-- result_t , xin, addout are UNSIGNED
with addsub select – combinational add/sub
      addout <=  (xin + result_t) when '1',
                 (xin - result_t)  when '0',
                 (others => '-')   when others;

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

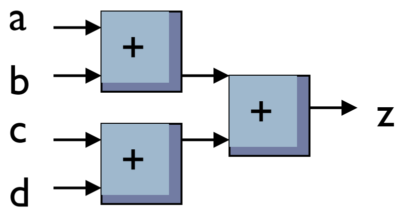
```
-- signal addsub eliminated – circuit will be the same
process (clr, clk) begin
        if (clr = '0') then
                result_t <= (others => '0');
        elsif rising_edge(clk) then
              case addsub is
                    when '1'      => result_t <=  (xin + result_t);
                    when '0'      => result_t <=  (xin - result_t);
                    when others => result_t <=  (others => '-');
              end case;
        end if;
 end process;
```

Multiple adder structures

```
z <= a + b + c + d;
-- 3 adders stacked 3 deep
```

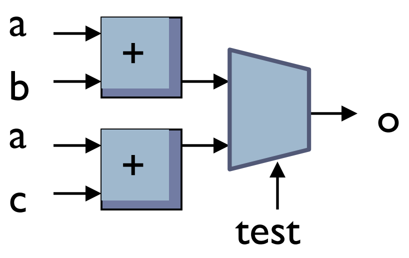


```
z <= (a + b) + (c + d);
-- 3 adders stacked 2 deep
```



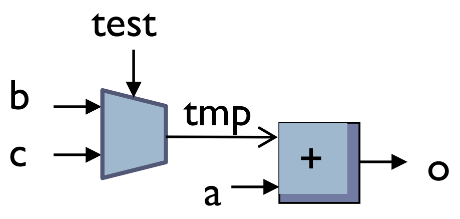
Resource sharing for mutually-exclusive operations

```
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
```


test
b
c
tmp
+
a
o