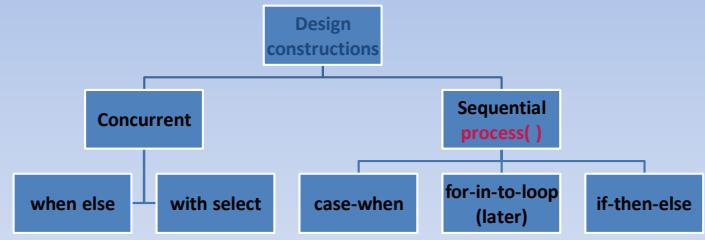
Välkommen till matematik avsnittet

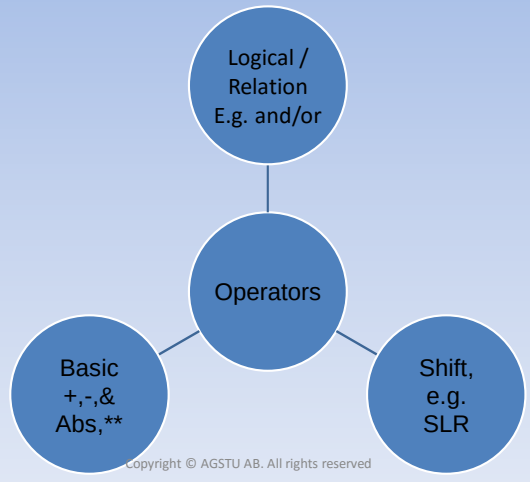
- Overview of operators
- Arithmetic operations
- Numbers
- VHDL Types & Packages
  - std_logic_arith and numeric_std

Version	Date	Responsible	Description
0.0	2011	LL & Mia	Preliminary version
1.0	2015	LL	Updated

Design constructions repeating



Typical Operators



AGSTU

Logical / relation operators

- Logical operators
  - (A nand B) nand C ≠ A nand (B nand C)
- Relation operators
  - = equal ; /= not equal
  - < <= > >= smaller, bigger , equal etc.

- NOT
- AND
- NAND
- OR
- NOR
- XOR
- EXOR

AGSTU

Shift operators

- Logical shift and rotate
  - Sll (shift left logical, fill blank with 0);
  - srl (shift right logical, fill blank with 0)
  - rol(rotate left logical); ror(rotate right logical) circular operation.
  - E.g. “10010101” rol 3 is “10101100”
- Arithmetic shift (http://en.wikipedia.org/wiki/Arithmetic_shift)
  - sla (shift left arithmetic) fill blank with 0,same as sll (shift left logical)
  - sra (shift right arithmetic), fill blank with sign bit (MSB)

Some basic operators

- **‘&’** concatenation: ‘0’ & ‘1’ is “01”, Notice the use of “&”.
- signal Z_BUS : bit_vector (3 downto 0); signal A_BIT, B_BIT, C_BIT, D_BIT : bit; begin Z_BUS <= A_BIT & B_BIT & C_BIT & D_BIT;
- byte <= a_bus & b_bus;
- byte <= a_bus & '0' & '0' & '0' & '1';

Arithmetic operations

Symbol	Function
+	addition
-	subtraction
*	multiplication
/	division
abs	absolute value
rem	rest
mod	modulus
**	exponent

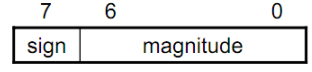
These arithmetic operations are predefined data types: integer and time.

Not for std_logic_vector. If you want to use std_logic_vector you must include Math package.

The basics of signed numbers

- **Integers** are formed by the natural numbers (including 0) (0, 1, 2, 3, ...) together with the negatives of the non-zero natural numbers (−1, −2, −3, ...).
- Positive numbers are called unsigned in VHDL
- How can we represent signed numbers? (+ and -)
  - Solution 1: **Sign-magnitude**
    - use one bit to represent the sign, and the remaining bits to represent the magnitude

sign = 0 → +ve
sign = 1 → -ve



+27 = 00011011
-27 = 10011011

- Problems:
  - need to handle sign and magnitude separately
  - two values for zero (e.g., 00000000, 10000000)
  - not convenient for arithmetic

Two's complement

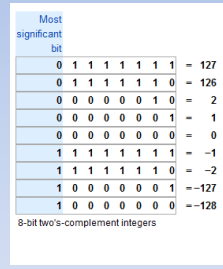
- Solution 2 is to represent negative numbers by taking the magnitude, inverting all bits, and adding one.

- This is called two's complement

Positive number	+27	=	0001 1011
Invert all bits			1110 0100
Add 1	-27	=	**1110 0101**

- Taking the two's complement again give the original number:

Negative number	-27	=	1110 0101
Invert all bits			0001 1010
Add 1	+27	=	**0001 1011**



Arithmetic operations

Synthesizable arithmetic operations:

- Addition, +
- Subtraction, -
- Comparisons, >, >=, <, <=
- Multiplication, *
- Division by a power of 2 (equivalent to shift)

Unsigned and signed types

- Unsigned type
  - Value 0 to 2^n-1
- Signed type
  - Value $-2^{(n-1)}$ to $2^{(n-1)}$ -1

More to read:
http://en.wikipedia.org/wiki/Two%27s_complement

- Usage similar to std_logic_vector:

```
signal A_unsigned    : unsigned(3 downto 0) ;
signal B_signed      : signed  (3 downto 0) ;
signal C_slv         : std_logic_vector (3 downto 0) ;
. . .
A_unsigned <= "1111" ;      <-- = 15 decimal
B_signed   <= "1111" ;      <-- = -1 decimal
C_slv      <= "1111" ;      <-- = 15 decimal only if using std_logic_unsigned
```


Adder with Carry Out

```
    '0',  A(3:0)
+   '0',  B(3:0)
------------------
CarryOut, Result(3:0)
```

```
Y5 <= ('0' & A) + ('0' & B);
Results <= Y5(3 downto 0) ;
CarryOut <= Y5(4) ;
```

Considerations

- **Type declaration**
  - **signal A8, B8, Result8 : unsigned(7 downto 0) ;**
  - **signal Result9 : unsigned(8 downto 0) ;**
  - **signal Result7 : unsigned(6 downto 0) ;**
- **-- Simple Addition, no carry out**
  - **Result8 <= A8 + B8 ;**
- **-- Carry Out in result**
  - **Result9 <= ('0' & A8) + ('0' & B8) ;**
- **-- For smaller result, slice input arrays**
  - **Result7 <= A8(6 downto 0) + B8(6 downto 0) ;**

AGSTU

 13

Considerations

- Synthesis tools create a 32-bit wide resource for unconstrained integers
  - **signal Y_int, A_int, B_int : integer ;**
  - **Y_int <= A_int + B_int ;**
- Do not use unconstrained integers for synthesis
  - **signal A_int, B_int: integer range -8 to 7;**
  - **signal Y_int : integer range -16 to 15 ;**
- **Recommendation:** Specify a range with integers

AGSTU

 14

Signed and unsigned

- **Recommendation:** Use std_logic or vectors for ports -> better control of your design,
- Use signed and unsigned inside the architecture,
- Result: More robust code.

AGSTU

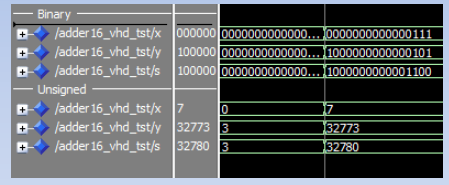
Packages for Numeric Operations

- **numeric_std -- IEEE standard**
  - **library ieee ;**
  - **use ieee.std_logic_1164.all ;**
  - **use ieee.numeric_std.all ;**
- **-- Synopsys,** a defacto industry standard
  - **library ieee ;**
  - **use ieee.std_logic_1164.all ;**
  - **use ieee.std_logic_arith.all ;**
  - **use ieee.std_logic_unsigned.all ; or IEEE.std_logic_signed.al**
- **Recommendation?:**
  - Use numeric_std for new designs?

AGSTU

Addition of Unsigned Numbers (1-3)

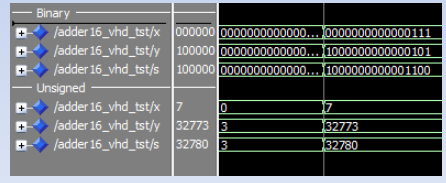
X <= "0000000000000000" after 100ns, "0000000000000111" after 200ns;
Y <= "0000000000000011" after 100ns, "1000000000000101" after 200ns;



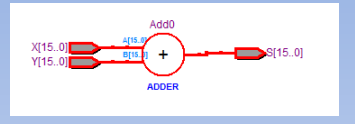
Total logic elements	16 / 114,480 (< 1 %)
Total combinational functions	16 / 114,480 (< 1 %)
Dedicated logic registers	0 / 114,480 (0 %)
Total registers	0

Addition of Unsigned Numbers

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;

ENTITY adder16 IS
      PORT (   X, Y          : IN     UNSIGNED(15 DOWNTO 0) ;
               S             : OUT    UNSIGNED (15 DOWNTO 0) ) ;
END adder16 ;


ARCHITECTURE Behavior OF adder16 IS

BEGIN
      Sum <= X + Y;
END Behavior ;
```

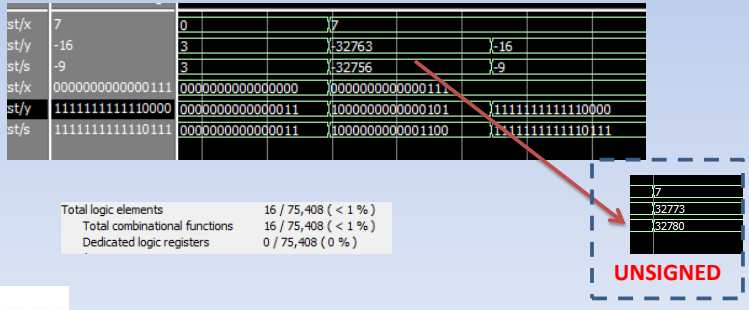


Signed



X <= "0000000000000000" after 100ns, "0000000000000111" after 200ns ;

Y <= "0000000000000011" after 100ns, "1000000000000101" after 200ns, "1111111111110000" after 250ns;

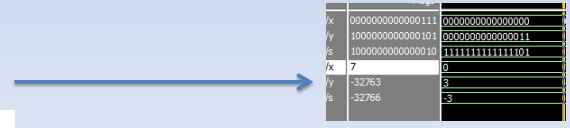


Total logic elements	16 / 75,408 (< 1 %)
Total combinational functions	16 / 75,408 (< 1 %)
Dedicated logic registers	0 / 75,408 (0 %)

Addition of Signed Numbers

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;

ENTITY adder16 IS
      PORT (   X, Y              : IN     SIGNED(15 DOWNTO 0) ;
               S                 : OUT    SIGNED(15 DOWNTO 0) ) ;
END adder16 ;
ARCHITECTURE Behavior OF adder16 IS
BEGIN
      S <= X + Y;
END Behavior ;
```

S <= X + Y;



Example: numeric_std, type conversion

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;

ENTITY adder16 IS
PORT (

X, Y : IN STD_LOGIC_VECTOR(15 DOWNTO 0) ;
S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;

END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
BEGIN

S <= std_logic_vector(unsigned(X) +
unsigned(Y));

END Behavior ;
```

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;

ENTITY adder16 IS
PORT (

X, Y : IN STD_LOGIC_VECTOR(15 DOWNTO 0) ;
S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;

END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
BEGIN

S <= std_logic_vector(signed(X) + signed(Y));

END Behavior ;
```


The old way? Example: std_logic_arith

Learn this also! In the book

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity adder is
 port(
   a : in STD_LOGIC_VECTOR(2 downto 0);
   b : in STD_LOGIC_VECTOR(2 downto 0);
   c : out STD_LOGIC_VECTOR(2 downto 0)   );
end adder;

architecture adder_arch of adder is
begin
 c <= a + b;
end adder_arch;
```

Tells compiler to treat std_logic_vector like unsigned type

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adder16 IS
PORT (

X, Y : IN STD_LOGIC_VECTOR(15 DOWNTO 0) ;
S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)) ;

END adder16 ;


ARCHITECTURE Behavior OF adder16 IS   BEGIN
        S <= X + Y ;
END Behavior ;
```

Tells compiler to treat std_logic_vector like signed type

Signed and unsigned numbers

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all ;

ENTITY adder16 IS
        PORT (  X, Y    : IN    SIGNED(15 DOWNTO 0) ;
                S       : OUT   SIGNED(15 DOWNTO 0) ) ;
END;

ENTITY adder16 IS
        PORT (  X, Y    : IN    UNSIGNED(15 DOWNTO 0) ;
                S       : OUT   UNSIGNED(15 DOWNTO 0) ) ;
END;
```

```
ARCHITECTURE Behavior OF
adder16 IS
BEGIN
        S <= X + Y;
END Behavior ;
```


Multiplication and Division

- **Type declaration**
  - **signal A_unsigned_vector, B_unsigned_vector : unsigned(7 downto 0) ;**
  - **signal Z_unsigned_vector : unsigned(15 downto 0) ;**
  - **Z_unsigned_vector <= A_unsigned_vector * B_unsigned_vector;**

Multiplication of signed and unsigned numbers

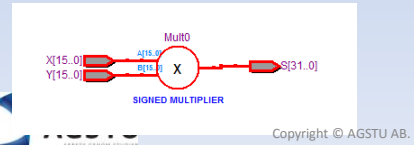
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all ;

ENTITY mult IS
        PORT (  X, Y    : IN    SIGNED(15 DOWNTO 0) ;
                S       : OUT   SIGNED(31 DOWNTO 0) ) ;
END;

ENTITY mult IS
        PORT (  X, Y    : IN    UNSIGNED(15 DOWNTO 0) ;
                S       : OUT   UNSIGNED(31 DOWNTO 0) ) ;
END;
```

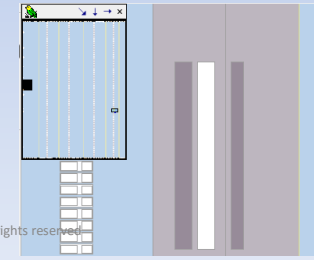
```
ARCHITECTURE Behavior OF mult
IS
BEGIN
        S <= X * Y;
END Behavior ;
```


Multiplication

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;

ENTITY adder16 IS
    PORT (  X, Y        : IN    SIGNED(15 DOWNTO 0) ;
            S           : OUT   SIGNED(31 DOWNTO 0) ) ;
END adder16 ;
ARCHITECTURE Behavior OF adder16 IS
BEGIN
    S <= X * Y;
END Behavior ;
```

Embedded Multiplier 9-bit elements	2 / 400 (< 1 %)





Multiplication of signed and unsigned numbers, type converting

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all ;

entity multiply is
        port(
                a : in STD_LOGIC_VECTOR(7 downto 0);
                b : in STD_LOGIC_VECTOR(7 downto 0);
                cu : out STD_LOGIC_VECTOR(15 downto 0);
                cs : out STD_LOGIC_VECTOR(15 downto 0)
        );
end multiply;

architecture dataflow of multiply is

SIGNAL sa: SIGNED(7 downto 0);
SIGNAL sb: SIGNED(7 downto 0);
SIGNAL sc: SIGNED(15 downto 0);

SIGNAL ua: UNSIGNED(7 downto 0);
SIGNAL ub: UNSIGNED(7 downto 0);
SIGNAL uc: UNSIGNED(15 downto 0);
```

```
begin

-- signed multiplication
      sa <= SIGNED(a);
      sb <= SIGNED(b);
      sc <= sa * sb;
      cs <= STD_LOGIC_VECTOR(sc);

-- unsigned multiplication
      ua <= UNSIGNED(a);
      ub <= UNSIGNED(b);
      uc <= ua * ub;
      cu <= STD_LOGIC_VECTOR(uc);

end dataflow;
```


Multiplication, Type converting

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;

ENTITY adder16 IS
        PORT (    X, Y                : IN      STD_LOGIC_VECTOR(15 DOWNTO 0) ;
                  S                   : OUT     STD_LOGIC_VECTOR(31 DOWNTO 0) ) ;
END adder16 ;
ARCHITECTURE Behavior OF adder16 IS
BEGIN
        S <= std_logic_vector(signed(X) * signed(Y));
END Behavior ;
```