ECE 2214

Digital Logic Design

L-17 Design of arithmetic using CAD tools

Fall 2022

Outline

- Schematic description of arithmetic logic circuits
- VHDL description of arithmetic logic circuits
  - Modules and packages
  - Number representations in VHDL
  - Arithmetic operators in VHDL
- Arithmetic assignment statements in VHDL

Summary of previous lecture

UNB

- Learned multiplication and division of binary numbers
- Fractional representation
  - Fix point representation

$$B = b_{n-1}b_{n-2}\cdots b_0 \bullet b_{-1}b_{-2}\cdots b_{-k}$$

  - Floating number representation

$$\mathbf{Value_{10} = \pm 1.M * R^{E\text{-}bias}}$$

Binary-coded-decimal representation (BCD)

- Use four bits to encode each decimal digits. Each digit is encoded using unsigned binary representation as:

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Note:

- Decimal digit (0…9) → only need 4 bits
- Only the first 10 of the 16 possible numbers obtained with 4 bits are used, the rest can't occur in BCD
- Benefit: provide a format that is easy to display
  - Not commonly used in current computers
- The arithmetic operation of BCD can be complicated
  - Any arithmetic result $>(9)_{10}$ must keep the excess magnitude of 9 and carry one to the next digit

- Example 6: Convert the following BCD number into binary

$$934 = \underset{\mathbf{9}}{1001}\ \ \underset{\mathbf{3}}{0011}\ \ \underset{\mathbf{4}}{0100}$$

ASCII Character code

- This is the standard code for storing text characters
  - Each letter and special character has a unique code
- It uses 7 bits to represent each character (127 in total)

- Example 7: Describe the relationship between decimal, ASCII and binary code for numbers from 0..9

$(0)_{10} = 30_{ASCII} = (0110000)_2$

$(1)_{10} = 31_{ASCII} = (0110001)_2$

$(2)_{10} = 32_{ASCII} = (0110010)_2$

$(3)_{10} = 33_{ASCII} = (0110011)_2$

$(4)_{10} = 34_{ASCII} = (0110100)_2$

$(5)_{10} = 35_{ASCII} = (0110101)_2$

$(6)_{10} = 36_{ASCII} = (0110110)_2$

$(7)_{10} = 37_{ASCII} = (0110111)_2$

$(8)_{10} = 38_{ASCII} = (0111000)_2$

$(9)_{10} = 39_{ASCII} = (0111001)_2$

See 5.4 pp. 302 of the textbook for a full list

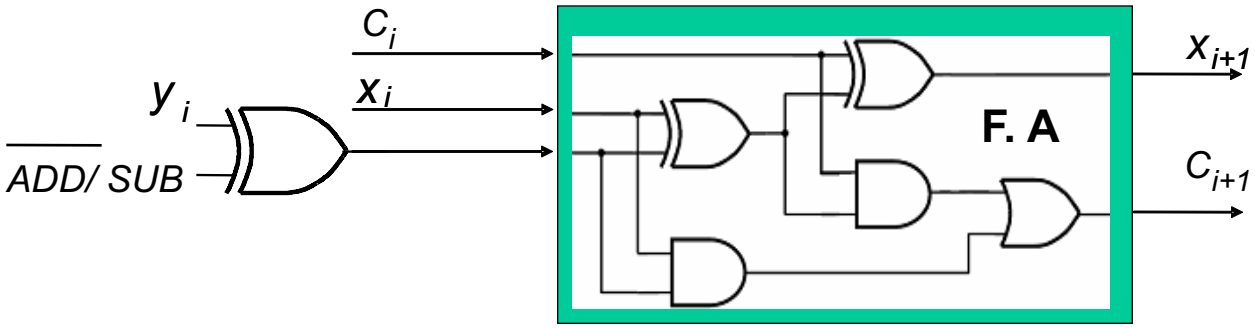
Design of arithmetics using CAD tools

UNB

- There are two approaches designing arithmetic circuits
  1. Schematic description
     - Involve providing the schematic that contains the logic gates performing the arithmetic
  2. Used HDL languages (VHDL)
     - Involve providing scripts that describe operation of the circuit that perform the arithmetic operation

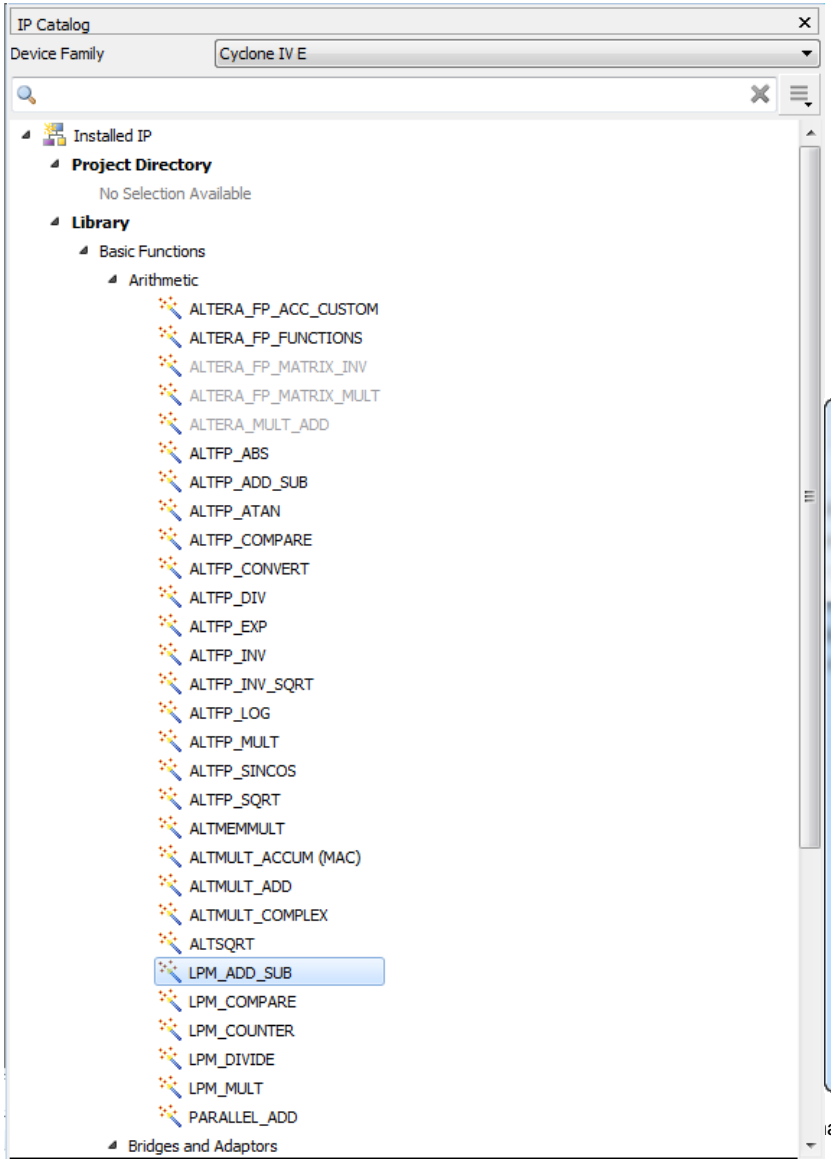
Schematic description of arithmetic circuits

- Higher level arithmetic has to be designed and wired graphically using hierarchical file structures
- Example: To create an n-bit adder you could design a one-bit full-adder module and connect **n** instances of the that module

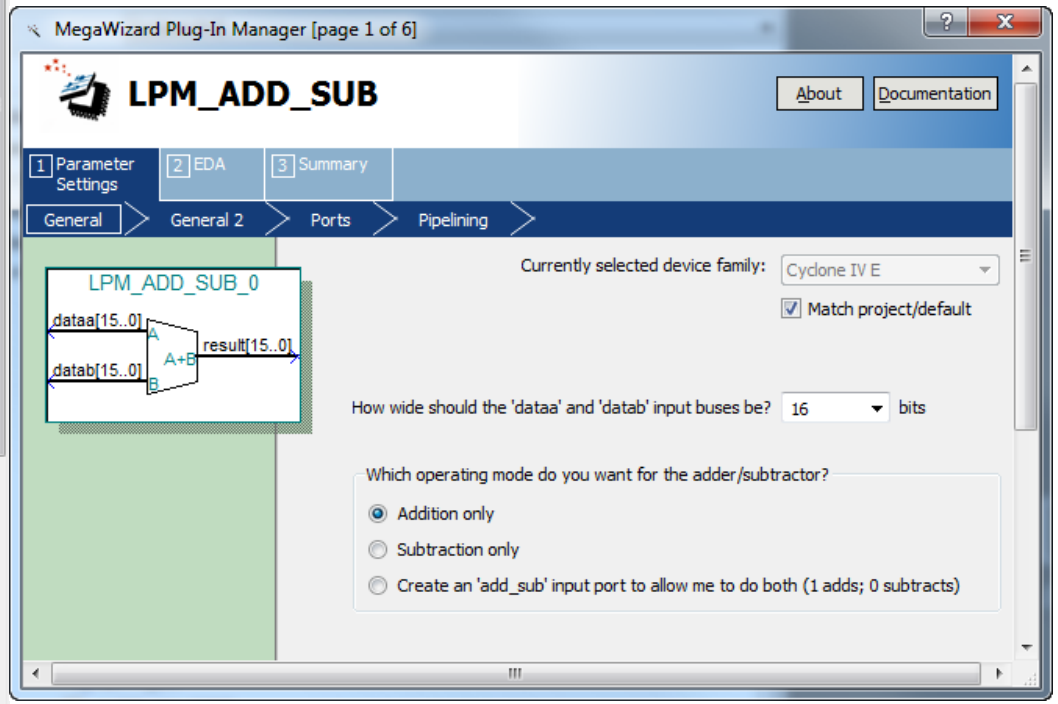


- More cumbersome, especially when the number of bits is high but…
- You can use standard modules available in Quartus library for most of the basic arithmetic operations
  - Some modules are technology dependent
  - Others are generic for any type of chip

Use of the schematic arithmetic library

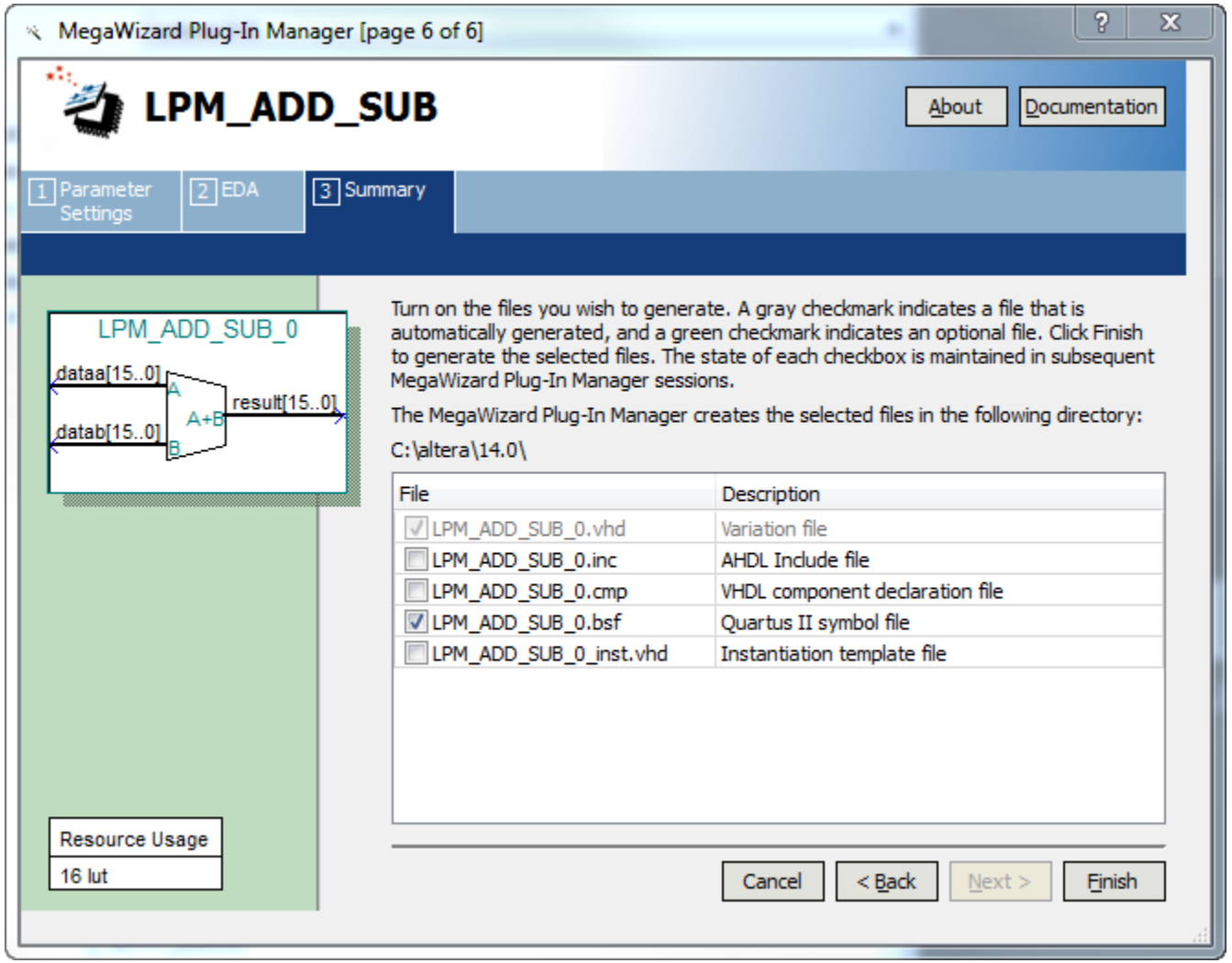


- Arithmetic library available from the menu of Quartus II
  - \Tools\IP Catalog
  - Several modules available
    - # of bits configurable!



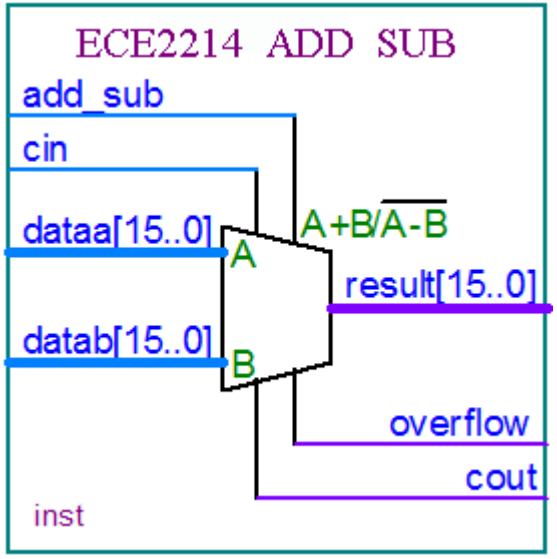
LPM library arithmetic library

- Can generate more than one format



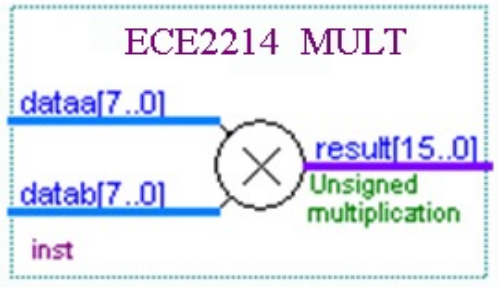
Use of the schematic arithmetic library

- Module generated with the IP Catalog
  - 16-bits adder
  - Carry-out
  - Overflow detection



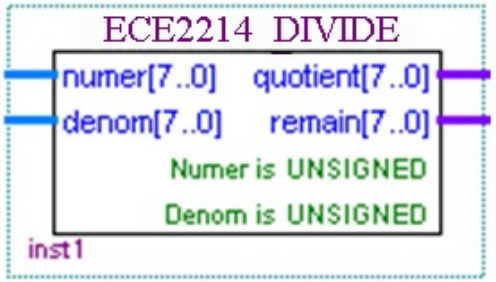
Use of the schematic arithmetic library

- IP Catalog library has other modules
- i.e. Multiplication
  - 8-bits multiplier
    - Two 8-bit input numbers
    - One 16 bit output number



Use of the schematic arithmetic library

- IP Catalog library has other modules
- i.e. division
  - Unsigned number divider
    - Two 8-bit input numbers
    - One 8-bit quotient output
    - One 8-bit remainder output



Arithmetic with HDL languages

- Involve the development of scripts that perform the arithmetic operations
- Scripts have to follow certain structure given by the language used
  - VHDL has a general script structure based on Entity-Architecture
  - i.e. Similar to the adder generated before but with only 1 bit

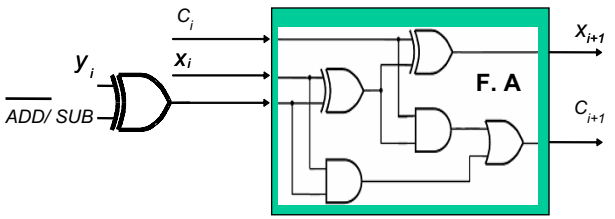
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

 ENTITY fulladd_L18_Slide11 IS
 PORT(Cin , x, y    : IN       STD_LOGIC;
          s, Cout      : OUT    STD_LOGIC);
END;

ARCHITECTURE logicFunc OF fulladd_L18_Slide11 IS
BEGIN
          s <= (x XOR y) XOR Cin ;
         Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y);
END LogicFunc;
```

$$S_i = x_i \oplus y_i \oplus c_i$$

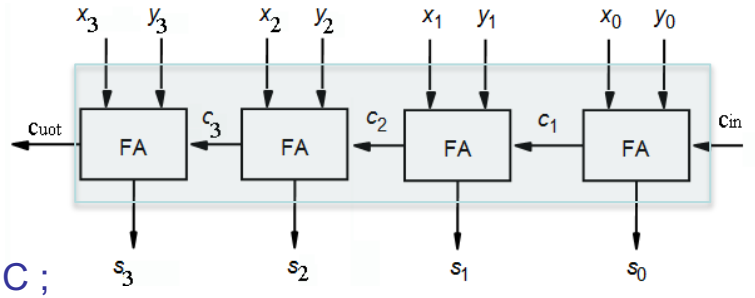
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$



- N-bit carry adder is the implemented instantiating this modules n-times

Arithmetic with VHDL language
Direct implementation with basic sentences

- Example of a 4-bit ripple carry adder




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY adder4_L18_slide12 IS
        PORT (    Cin : IN STD_LOGIC ;
                  x3, x2, x1, x0          : IN STD_LOGIC ;
                  y3, y2, y1, y0          : IN STD_LOGIC ;
                  s3, s2, s1, s0          : OUT STD_LOGIC := '0' ;
                  Cout                    : OUT STD_LOGIC ) ;
END adder4_L18_slide12 ;
ARCHITECTURE Structure OF adder4_L18_slide12 IS
   SIGNAL c1, c2, c3 : STD_LOGIC ;
   COMPONENT fulladd_L18_Slide11
     PORT(Cin , x, y    : IN     STD_LOGIC;
          s, Cout       : OUT    STD_LOGIC);
   END COMPONENT ;
BEGIN
   stage0: fulladd_L18_Slide11 PORT MAP ( Cin, x0, y0, s0, c1 ) ;
   stage1: fulladd_L18_Slide11 PORT MAP ( c1, x1, y1, s1, c2 ) ;
   stage2: fulladd_L18_Slide11 PORT MAP ( c2, x2, y2, s2, c3 ) ;
   stage3: fulladd_L18_Slide11 PORT MAP (c3, x3, y3, s3,  Cout) ;
END Structure ;
```

A file must be saved with the same name: Fulladd_L18_Slide11.vhd

- Component structure used to instantiate fulladd module 4 times

Arithmetic with VHDL language
Direct implementation with basic sentences

UNB

- Observations
  - Four bits are declared for the inputs ***x*** and ***y*** and for the output ***s***
  - A separate bit is declared for the ***Cout*** and ***Cin***
  - The code that perform the full adder “fulladd” is declared as a COMPONENT
  - The typical declaration of a component is:

```
COMPONENT name
        PORT ( in1, in2, … inN     : IN DataType ;
                    out1, out2, … outN: OUT DataType  ) ;
END COMPONENT ;
```

  - Name of component function must match the filename containing it
  - Internal variables or signals can be declared to store temporary results
    - Such as c1, c2 and c3
  - The declaration for the internal variables is:

```
SIGNAL vname1, vname2 … vnameN : DataType ;
```

Arithmetic with VHDL language
Direct implementation with basic sentences

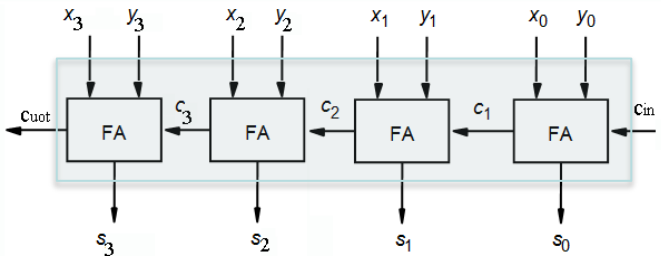
- Same function can be implemented with several approaches
- i.e. The same previous example can be written with an alternative style
  - Use VHDL package sentence instead of the component used before
  - A library style

The file fulladd_package.vhd must exist. It can have the implementation of fulladd_L18_Slide11 or use it from the separate file

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

PACKAGE fulladd_package IS
        COMPONENT fulladd_L18_Slide11
                PORT ( Cin, x, y      : IN STD_LOGIC ;
                             s, Cout  : OUT STD_LOGIC ) ;
        END COMPONENT ;
END fulladd_package ;
```

HDL languages
Direct implementation with basic sentences




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.fulladd_package.all;

ENTITY fulladd_L18_Slide15 IS
        PORT (  Cin             : IN    STD_LOGIC ;
                x3, x2, x1, x0  : IN    STD_LOGIC ;
                y3, y2, y1, y0  : IN    STD_LOGIC ;
                s3, s2, s1, s0  : OUT   STD_LOGIC ;
                Cout            : OUT   STD_LOGIC ) ;
END fulladd_L18_Slide15 ;

ARCHITECTURE Structure OF fulladd_L18_Slide15 IS
        SIGNAL c1, c2, c3 : STD_LOGIC ;
BEGIN
        stage0: fulladd_L18_Slide11 PORT MAP ( Cin, x0, y0, s0, c1 ) ;
        stage1: fulladd_L18_Slide11 PORT MAP ( c1, x1, y1, s1, c2 ) ;
        stage2: fulladd_L18_Slide11 PORT MAP ( c2, x2, y2, s2, c3 ) ;
        stage3: fulladd_L18_Slide11 PORT MAP ( c3, x3, y3, s3, Cout) ;
END Structure ;
```

HDL languages
Direct implementation with basic sentences

UNB

- The package is a more organized style
  - Structure the 1-bit adder as a package
  - The package is saved as a library (work.fulladd_package.all in this case)
  - The library can be stored in a separate file in the same work directory

  - The 4-bit adder script instantiate the 1-bit adder 4 times

Both code styles do the same operation
You can use the one that you prefer!

Number representation in VHDL
Basic number formats

- VHDL has different data objects to represent individual logic signals or wires
- A signal has to be declared with type associated in the following format:

SIGNAL signal_name : type_name;

- The type_name determines the values that the signal can have and … its use in VHDL
- The most common signal types are:
  - BIT
  - BIT_VECTOR
  - STD_LOGIC
  - STD_LOGIC_VECTOR
  - INTEGER (SIGNED or UNSIGNED)
  - BOOLEAN
  - ENUMERATION

Number representations in VHDL
Basic number formats

- **BIT or BIT_VECTOR:** Are predefined VHDL types to define the number of bits of a signal

BIT: Represent one bit and can have value '0' or '1'
BIT_VECTOR: Represent a linear array of BITS

Use: "TO' or "DOWNTO" specify the bit range

- Example:

```
SIGNAL  X       : BIT;
SIGNAL  X       : BIT_VECTOR (1 TO 4);
SIGNAL  X       : BIT_VECTOR (7 DOWNTO 0);
```

Note:

- TO indicate the direction from MSB (1) to LSB (4)
- DOWNTO indicate the direction from MSB (7) to LSB(0)
- Special care has to be taken in this definition since the interpretation of the result depends on the bit definition

Number representations in VHDL
Basic number formats

UNB

- **STD_LOGIC or STD_LOGIC_VECTOR:** Added to VHDL by IEEE 1164 standard library
  - Provides more flexibility than the BIT type
    - Store standard logic data such as bits and vectors
- Requires the term “USE ieee.std_logic_signed” for signed numbers and … the term “USE ieee.std_logic_unsigned” for unsigned numbers in the script heading

STD_LOGIC: Can hold different values ‘0’, ‘1’, ‘Z’(high impedance), ’-‘ (don’t care)

STD_LOGIC_VECTOR: Represent a linear array of STD_LOGIC

Use ‘TO’ or DOWNTO to specify the limits of the bits that the object holds

- Example:

```
SIGNAL  X   : STD_LOGIC;
SIGNAL  X   : STD_LOGIC_VECTOR (1 TO 4);
SIGNAL  X   : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

Number representations in VHDL
Basic number formats

- **INTEGER:** It is part of the standard VHDL code to represent signed numbers
  - Is designed to be use with arithmetic operators
- It has 32 bits by default and can represent number from $-(2^{31}-1)$ to $(2^{31}-1)$.
- Integer with fewer bits can be declared using the keyword RANGE

INTEGER: Represent a 32-bit integer type

INTEGER RANGE $-L_1$ TO L_2

Represent an integer number with fewer bits than 32 determined by L_1 and L_2

Example:

```
SIGNAL  X :  INTEGER;
SIGNAL  X : INTEGER RANGE -127 TO 127; (8 Bits)
```

- **BOOLEAN:** Only have TRUE (‘1’) or FALSE (‘0’) values
- Example:

```
SIGNAL  X      : Boolean;
```

Number representations in VHDL
Basic number formats

UNB

- **ENUMERATION:** Signal can only have one of the user defined values
  - Signal is declared as the user defined type

    TYPE enumeration_type_name **IS** (name1, name2, … nameN);

    SIGNAL X : enumeration_type_name;

  Example:

    TYPE State_Type IS (stateA, stateB, stateC);
    SIGNAL **y** : State_Type;

- **Array:** used with custom-defined data type specified before

  TYPE array_type_name **IS ARRAY** (index_range) OF element_type;

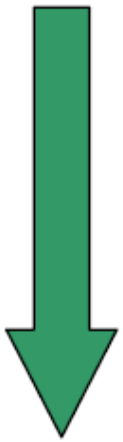
  SIGNAL array_name : array_type_name [**:=** initial_values];

Example:

    TYPE Int4 IS ARRAY (1 to 4) OF integer;
    SIGNAL **x** : Int4;

Arithmetic operators in VHDL
Operators & precedence

- A number of operators are defined in VHDL to perform arithmetic operations (**requires the use of IEEE library**)
- The arithmetic operators are highlighted



1. Binary logical operators: **AND, OR, NAND, NOR, XOR, XNOR**
2. Relational operators: **=, /=, <, <=, >, >=**
3. Shift operators: **sll, srl, sla, sra, rol, ror**
4. Adding operators: **+, -, & (concatenation)**
5. Unary sign operators: +, -,
6. Multiplying operators: ***, /, mod, rem**
7. Miscellaneous operators: **not abs** **

Note:

- Operators of the same precedence will be evaluated from left to right.
- Bracket will change the precedence of operators.

X AND **Y** OR **Z** AND **M** is not XY + ZM → (**X** AND **Y**) OR (**Z** AND **M**)

Statements for arithmetic assignment in VHDL
Assignment of variables

The assignment in VHDL is accomplished as follow:

```
signal_name <= expression;
```

i.e If the following signals are defined

```
SIGNAL  X,Y,S       : STD_LOGIC_VECTOR_UNSIGNED (15 DOWNTO 0)
```

Addition operation statement : S <= X + Y; (16-bit adder)

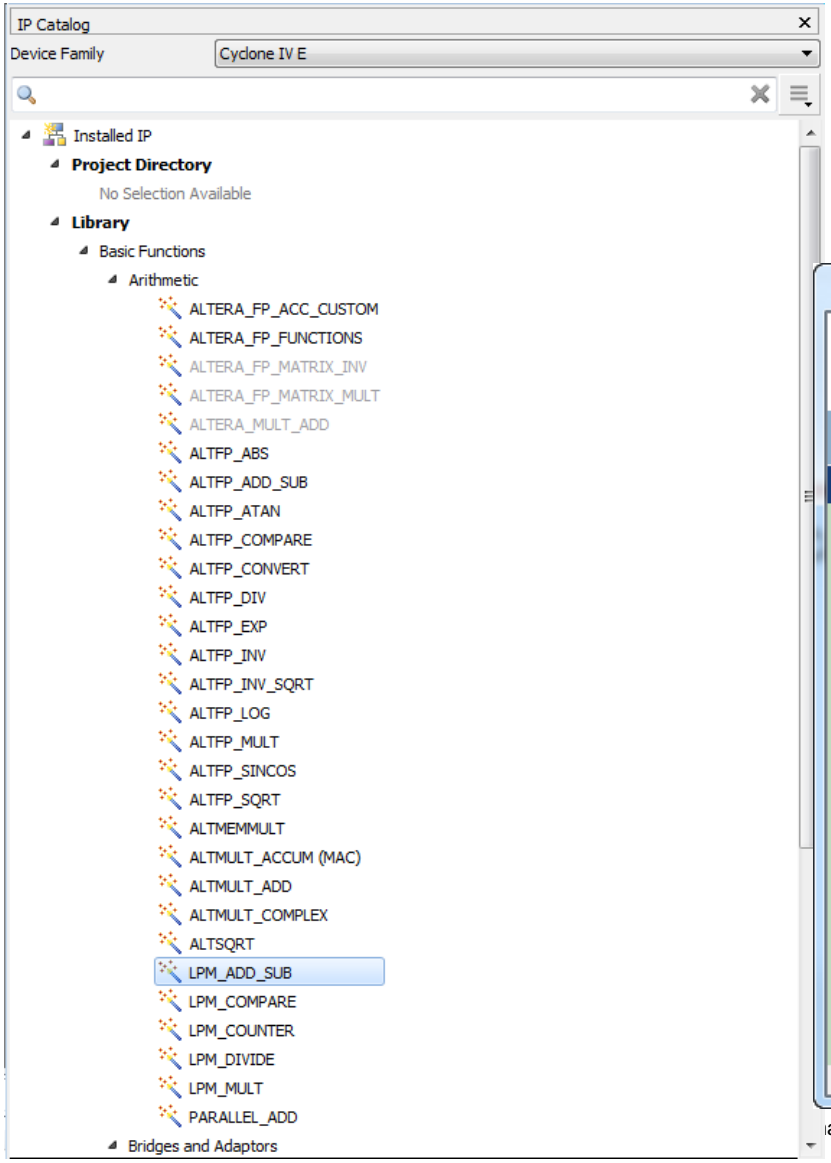
Multiplication operation statement : S <= X * Y; (16-bit multiplier)

A special and very important case is assigning constants to variables. In VHDL you can assign values to bits separately or in groups:

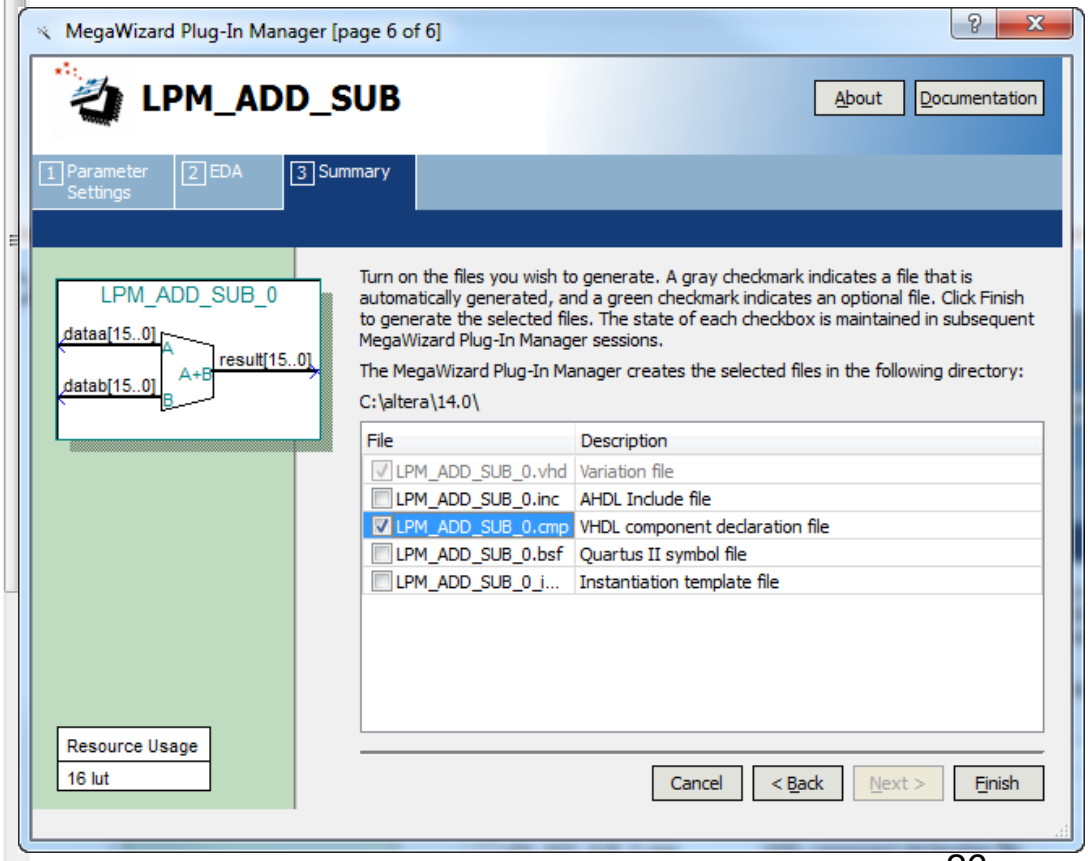
signal_name <= 'C'; → simple assignment of constant C to signal_name
i.e. X <= '1';

signal_name <= "$C_0C_1\ C_2 \ldots C_{N-1}C_N$"; Multiple assignment of constants C to signal_name. i.e. X <= "1001".

Use of the VHDL pre-built arithmetic library



- Similar to the schematic arithmetic library, you can generate arithmetic templates
  - \Tools\IP Catalog
  - Several configurable modules



Arithmetic with VHDL: QII LPM library

- Files generated:
  - L4_4_Bit_Adder.qip
  - L4_4_Bit_Adder.vhd (fraction of the code below)

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

LIBRARY lpm;
USE lpm.all;

ENTITY L4_4_Bit_Adder IS
    PORT
    (
       dataa     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
       datab     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
       cout      : OUT STD_LOGIC ;
       overflow    : OUT STD_LOGIC ;
       result      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
    );
END L4_4_Bit_Adder;

ARCHITECTURE SYN OF l4_4_bit_adder IS

    SIGNAL sub_wire0  : STD_LOGIC ;
    SIGNAL sub_wire1  : STD_LOGIC ;
    SIGNAL sub_wire2  : STD_LOGIC_VECTOR (3 DOWNTO 0);
```

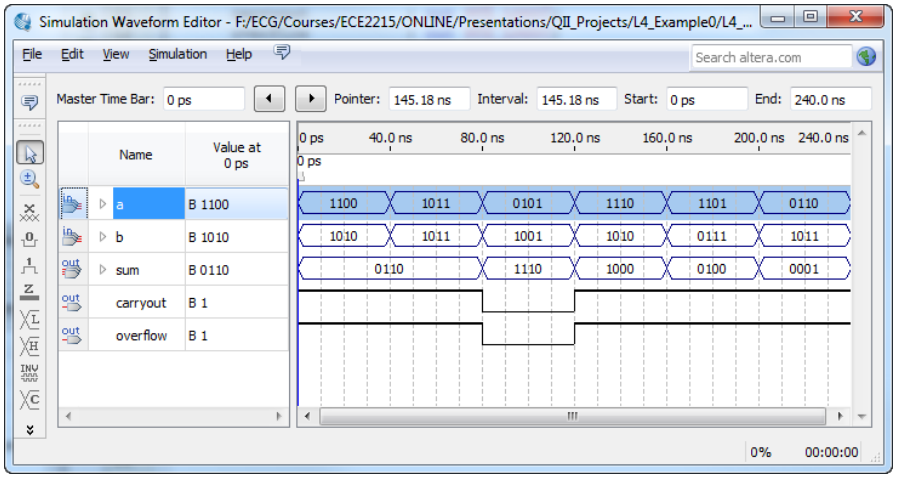
- Use component to reuse the VHDL code generated
- Similarly you can generate multiplier, divider circuits

Arithmetic with VHDL: QII LPM library

- 4-bit adder implementation

```
1  -- Quartus II VHDL- Unsigned 4-bit carry adder
2  -- ECE2215 L4
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6
7  entity L4_4_BIT_ADDER is
8     port (
9        a                 : in STD_LOGIC_VECTOR (3 downto 0);
10       b                 : in STD_LOGIC_VECTOR (3 downto 0);
11       carryout          : out STD_LOGIC;
12       overflow          : out STD_LOGIC;
13       sum               : out STD_LOGIC_VECTOR (3 downto 0)
14    );
15 end entity;
16
17 architecture rtl of L4_4_BIT_ADDER is
18  COMPONENT FOUR_BIT_ADDER IS
19     PORT
20     (
21        dataa     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
22        datab     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
23        cout      : OUT STD_LOGIC ;
24        overflow     : OUT STD_LOGIC ;
25        result      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
26     );
27     END COMPONENT;
28
29 begin
30
31    I1: FOUR_BIT_ADDER PORT MAP (a, b,carryout,overflow, sum);
32
33 end rtl;
```

Simulation results



Arithmetic with VHDL
IEEE Numeric library

UNB

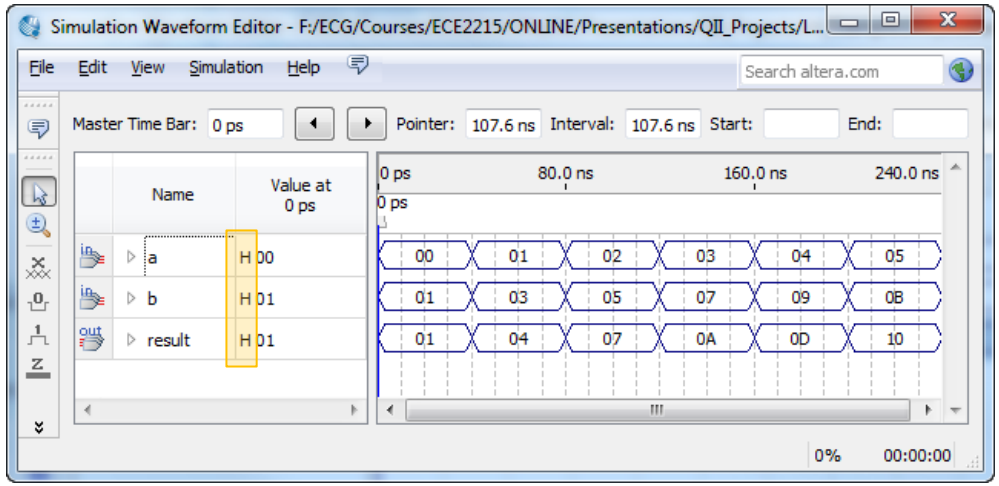
- This is the easiest approach
- Must include the library in your design as “use ieee.numeric_std.all;”
- Includes operators for signed and unsigned arithmetic
  - Automatically select the right operation based on type of operands
  - Supported operators: ‘+’, ‘-’, ‘*’, ‘/’, ‘rem’, ‘mod’,’**’, ‘abs’
  - Other comparison and converting functions as seen in:

    http://www.csee.umbc.edu/portal/help/VHDL/packages/numeric_std.vhd
  - Also support shifting operations basic gates with signed/unsigned types

Arithmetic with VHDL
IEEE Numeric library

- Example 1: 8-bit unsigned adder

```
1   -- L4 Example 1- Eight-bit unsigned adder
2   -- ECG 2016
3
4   library ieee;
5   use ieee.std_logic_1164.all;
6   use ieee.numeric_std.all;
7
8   entity L4_8bit_unsigned_adder is
9       generic
10      (
11          DATA_WIDTH : natural := 8
12      );
13      port
14      (
15          a      : in unsigned  ((DATA_WIDTH-1) downto 0);
16          b      : in unsigned  ((DATA_WIDTH-1) downto 0);
17          result : out unsigned ((DATA_WIDTH-1) downto 0)
18      );
19
20  end entity;
21
22  architecture rtl of L4_8bit_unsigned_adder is
23  begin
24
25      result <= a + b;
26
27  end rtl;
28
```



- All signed numbers are represented in hexadecimal as indicated by the H besides the variable name

Summary

UNB

- We introduced the design of arithmetic using CAD tools.
- There are two options:
  - Graphical description of arithmetic logic circuits
  - VHDL description of arithmetic logic circuits
- Both cases have predefined modules in the arithmetic library
  - \Tools\Mega Wizard Plug-in Manager
- We introduced all necessary information regarding VHDL to design arithmetic statements that work with signed and unsigned number representation

- References
  - Fundamentals of Digital Logic with VHDL Design 2nd Edition Stephen Brown, Zvonko Vranesic; McGraw-Hill, 2005. Chapter 5, pp. 278-289
  - Digital Design; Principles and Practices. Fourth Edition. John F. Wakerly, Prentice Hall, 2006. ISBN 0-13-186389-4. Chapter 5, pp. 256-289