ECE2215

Digital Logic Design Lab

Lab 9- Tutorial on Arithmetic with VHDL

1

Outline

- Arithmetic operators in VHDL
- Arithmetic with QII libraries
  - Signed and floating-point numbers
- Arithmetic with Numeric Standard Library
  - Signed and unsigned numbers

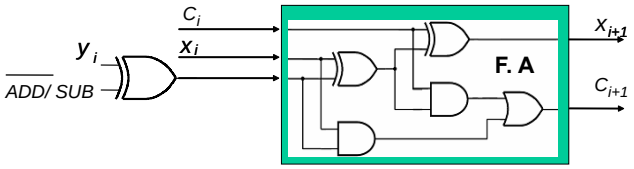
2

Design of arithmetic using CAD tools

- There are two approaches designing arithmetic circuits
  1. Schematic description
     - Involve providing the schematic that contains the logic gates performing the arithmetic
     - Not attractive for complex circuits
  2. Used HDL languages (VHDL)
     - Involve providing scripts that describe operation of the circuit that perform the arithmetic operation
     - Three alternatives
       1. Use default sentences in VHDL to implement the arithmetic
       2. Use QII libraries (i.e. LPM library)
       3. Use IEEE standard libraries (i.e. ieee.numeric_std library)

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 II library for most of the basic arithmetic operations
  - Some modules are technology dependent
  - Others are generic for any type of chip

Arithmetic with VHDL: Basic operators

- Involve the development of scripts that perform the arithmetic operations
- Scripts have to construct the logic functions required for the operation
  i.e. the addition of 1-bit is implemented with the equations:

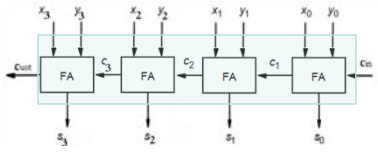
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fulladd_L4_Slide7 IS
PORT(Cin , x, y    : IN       STD_LOGIC;
        s, Cout      : OUT    STD_LOGIC);
END fulladd_L18_Slide11;

ARCHITECTURE logicFunc OF fulladd_L4_Slide7 IS
BEGIN
        s <= x XOR y XOR Cin ;
        Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y);
END LogicFunc;
```

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

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

Arithmetic with VHDL: Basic operators

- N-bit carry adder is the implemented instantiating this modules n-times
  - i.e. a 4-bit carry adders can be implemented as




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY adder4_L4_slide7 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_L4_slide7 IS
   SIGNAL c1, c2, c3 : STD_LOGIC ;
   COMPONENT fulladd_L4_Slide7 IS
     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 ;
```

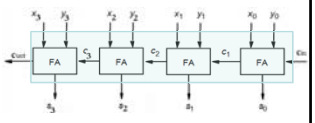
A file must be saved with the same name: fulladd_L4_Slide7.vhd

Arithmetic with VHDL: Custom library

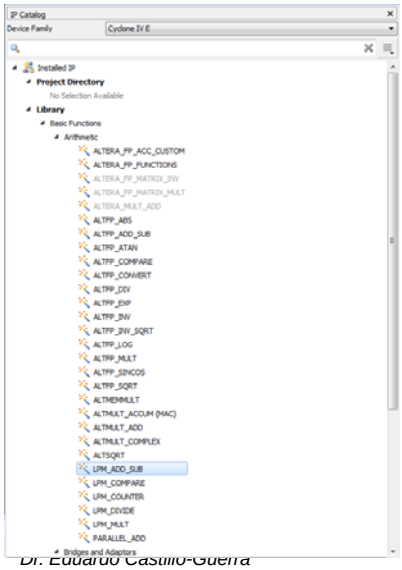
- Using PAKAGE to store a previous code in a custom library
  - work.fulladd_package.all contains the package definition of fulladd_L4_Slide7.vhd

```
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_L4_Slide7 PORT MAP ( Cin, x0, y0, s0, c1 ) ;
        stage1: fulladd_L4_Slide7 PORT MAP ( c1, x1, y1, s1, c2 ) ;
        stage2: fulladd_L4_Slide7 PORT MAP ( c2, x2, y2, s2, c3 ) ;
        stage3: fulladd_L4_Slide7 PORT MAP ( c3, x3, y3, s3, Cout) ;
END Structure ;
```

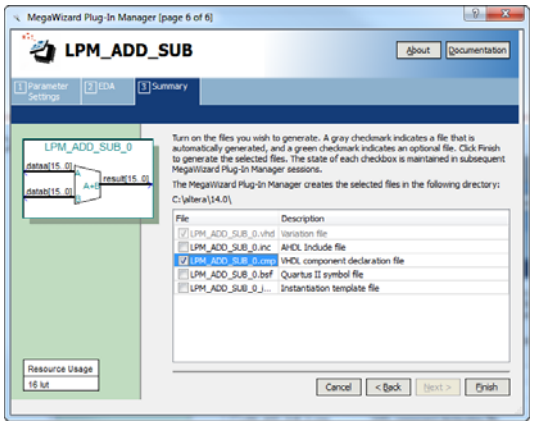
Indicates that the module fulladd_L4_Slide7 is in this library.



Arithmetic with VHDL: QII LPM library

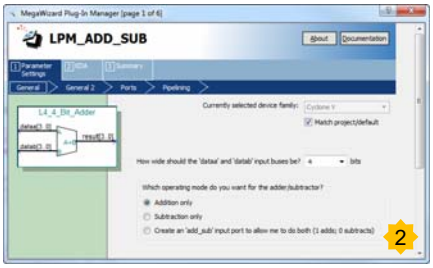


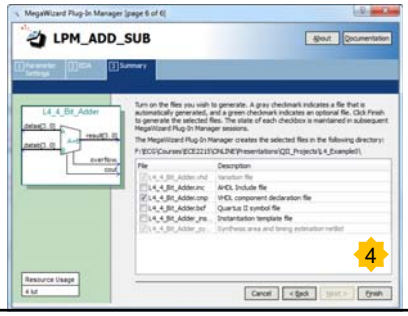
- You can generate arithmetic VHDL templates
  - \Tools\IP Catalog
  - Several configurable modules



Arithmetic with VHDL: QII LPM library

- Example 1 for the 4-bit carry adder
  - Creating the VHDL with LPM library





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);
```

- Generated files must be added to the project and placed in the project root directory
- Use component to reuse the VHDL code generated

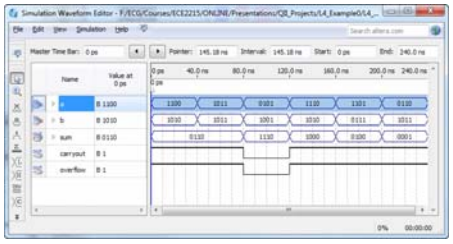
You can generate circuits to multiply and divide in a similar way

Arithmetic with VHDL: QII LPM library

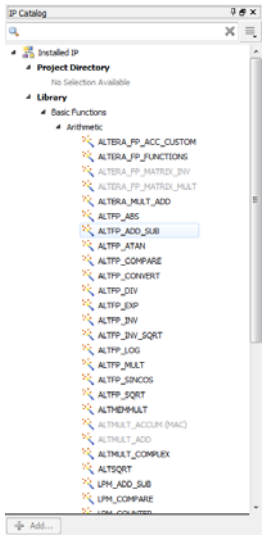
- 4-bit adder implementation

```
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;
```

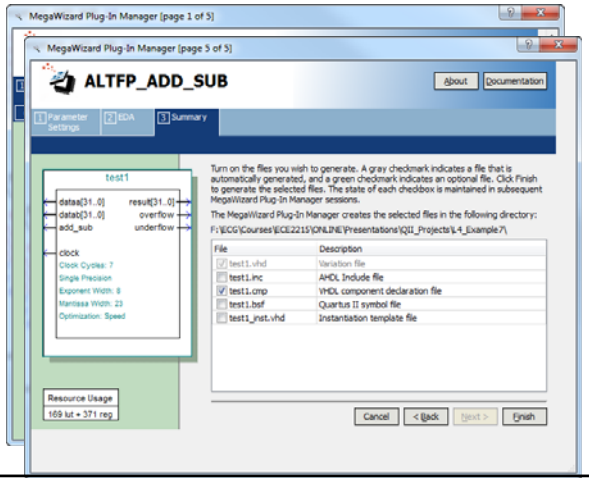
Simulation results



Arithmetic with VHDL: QII ALTFP library



- Another VHDL library for floating point numbers
  - Several other functions from "\Tools\IP Catalog"
  - Addition/subtraction function



Arithmetic with VHDL: QII ALTFP library

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

```
4860    LIBRARY ieee;
4861    USE ieee.std_logic_1164.all;
4862
4863    ENTITY FP_add_sub_function IS
4864       PORT
4865       (
4866          add_sub     : IN STD_LOGIC ;
4867          clock       : IN STD_LOGIC ;
4868          dataa       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
4869          datab       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
4870          overflow    : OUT STD_LOGIC ;
4871          result      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
4872          underflow   : OUT STD_LOGIC
4873       );
4874    END FP_add_sub_function;
4875
4876
4877    ARCHITECTURE RTL OF fp_add_sub_function IS
4878
4879       SIGNAL sub_wire0  : STD_LOGIC ;
4880       SIGNAL sub_wire1  : STD_LOGIC_VECTOR (31 DOWNTO 0);
4881       SIGNAL sub_wire2  : STD_LOGIC ;
```

- Use component to reuse the VHDL code generated
- VHDL code generated has many different functions in the top part
  - scroll down to find main function
  - same name you provided earlier
- Use the function definition to declare the component and “port map” your local variables

Arithmetic with VHDL: QII ALTFP library

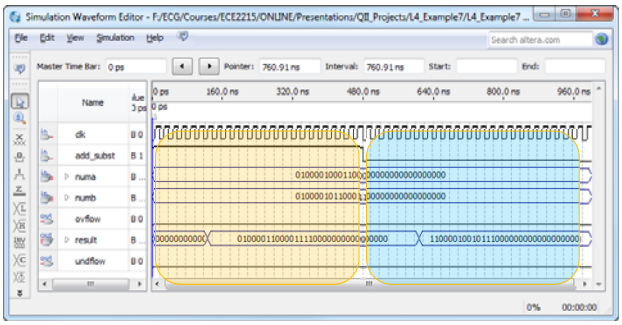
- Single precision floating point adder/substractor implementation

```
1   -- L4 Example 7- Floating Point add_sub with LPM library
2   -- ECE2215
3
4   library ieee;
5   use ieee.std_logic_1164.all;
6
7   entity L4_FP_single_precision_add_sub is
8      port
9      (  add_subst   : IN STD_LOGIC ;
10        clk         : IN STD_LOGIC ;
11        numa        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
12        numb        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
13        ovflow      : OUT STD_LOGIC ;
14        result      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
15        undflow     : OUT STD_LOGIC
16     );
17
18  end entity;
19
20  architecture rtl of L4_FP_single_precision_add_sub is
21     COMPONENT FP_add_sub_function
22     PORT (
23        add_sub     : IN STD_LOGIC ;
24        clock     : IN STD_LOGIC ;
25        dataa     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
26        datab     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
27        overflow    : OUT STD_LOGIC ;
28        result      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
29        underflow     : OUT STD_LOGIC );
30     END COMPONENT;
31
32  begin
33
34     I1: FP_add_sub_function port map (add_subst,clk,numa,numb,ovflow,result,undflow);
35
36  end rtl;
```

Arithmetic with VHDL: QII ALTFP library

- Single precision floating point adder/substractor implementation

Simulation results



- Added 44_{10} and 99_{10} resulting 143_{10}.
- Subtracted 99_{10} from 44_{10} resulting -55_{10}.
- A latency of 7 (7 clock cycle delay) to get result
- No overflow/underflow is generated

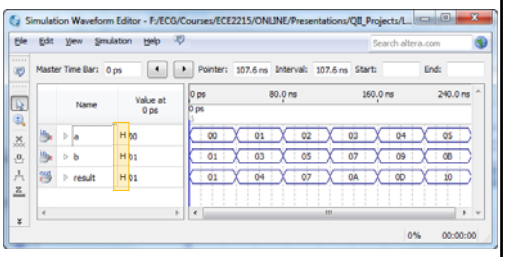
Arithmetic with VHDL: IEEE Numeric library

- 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
```

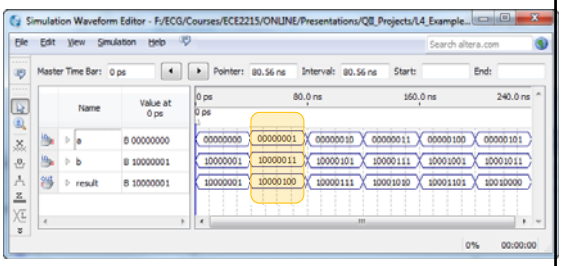


- All unsigned numbers are represented in hexadecimal as indicated by the H besides the variable's name

Arithmetic with VHDL: IEEE Numeric library

- Example 1: 8-bit signed adder

```
1  -- L4 Example- 8-bit Signed 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_signed_adder is
9
10    generic
11    (
12       DATA_WIDTH : natural := 8
13    );
14    port
15    (
16       a      : in signed ((DATA_WIDTH-1) downto 0);
17       b      : in signed ((DATA_WIDTH-1) downto 0);
18       result : out signed ((DATA_WIDTH-1) downto 0)
19    );
20 end entity;
21
22 architecture rtl of L4_8bit_signed_adder is
23 begin
24
25    result <= a + b;
26
27 end rtl;
```

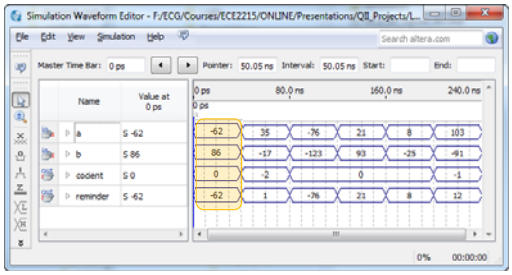


- All signed numbers are represented in 2's complement
  - 40-80ns→ 00000001 (+ $1_{2's}$)
    40-80ns→ + 10000011 (- $125_{2's}$)
    result 10000100 (- $124_{2's}$)

Arithmetic with VHDL: IEEE Numeric library

- Example 1: 8-bit signed division

```
1   -- L4 Example 6- Signed Divider
2   -- Lecture 4 ECE2215 "/" and "rem" operators
3
4   library ieee;
5   use ieee.std_logic_1164.all;
6   use ieee.numeric_std.all;
7
8   entity L4_8_bit_signed_div is
9       generic
10      (
11          DATA_WIDTH : natural := 8
12      );
13      port
14      (
15          a       : in signed ((DATA_WIDTH-1) downto 0);
16          b       : in signed ((DATA_WIDTH-1) downto 0);
17          cocient  : out signed ((DATA_WIDTH-1) downto 0);
18          reminder  : out signed ((DATA_WIDTH-1) downto 0)
19      );
20
21  end entity;
22
23  architecture rtl of L4_8_bit_signed_div is
24  begin
25
26      cocient <= a / b;
27      reminder <= a rem b;
28  end rtl;
```



NOTE: Numbers were formatted as Signed decimal in the simulation window to facilitate the verification of the performed operations. i.e.

First set of testing numbers:

num1→ -62_{10}= $11000010_{2's}$

num2→ $+86_{10}$= $01010110_{2's}$

cocient → 0_{10}= $11000010_{2's}$

reminder → -62_{10}= $11000010_{2's}$

Summary

- Arithmetic operators in VHDL
- Arithmetic with QII libraries
  - Signed and floating-point numbers
- Arithmetic with Numeric standard library
  - Signed and unsigned numbers

- Final exam recommendation
  - Bring code and Pin assignment from both boards
  - Bring any printed material you want
  - Will not have access to internet or online drives
  - Will be seated randomly
  - Practice bringing code compiled in other computers than the lab ones