VHDL Arithmetic and ALU

Announcements

- Homework #4 due Today
- Homework #5 posted
- Reading Assignment
  - Ch. 3 sections 7 & 18, Ch. 5 section 7

Numerical Representations

- Numbers in VHDL can be represented two different ways
  - STD_LOGIC_VECTOR
    - Array of bits
    - Treated as a binary number
    - Ex: X <= “01101100”, evaluated as 108
    - Industry standard and used in this class
  - Integer
    - Use sparingly (**and not in this class**)
    - Synthesis tool will create a 32-bit resource

Standard_Logic_Vector vs. Integer

```
architecture behave of traditionalcounter is
  signal int_count: std_logic_vector(7 downto 0);
begin
  main: process(clk, reset_n)
    begin
      if (reset_n = '0') then
          int_count <= (others => '0');
      elsif (clk'event and clk='1') then
          int_count <= int_count + 1;
      end if;
    end process;
  count <= int_count;
end behave;
```

```
architecture behave of traditionalcounter is
signal int_count: integer;
begin
main: process(clk, reset_n)
   begin
      if (reset_n = '0') then
            int_count <= 0;
      elsif (clk'event and clk='1') then
            int_count <= int_count + 1;
      end if;
   end process;
count <= int_count;
end behave;
```

Flow Status	Successful - Tue Sep 19 12:38:47 2017
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Standard Edition
Revision Name	counter
Top-level Entity Name	traditionalcounter
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	5 / 56,480 (< 1 %)
Total registers	9
Total pins	10 / 268 (4 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Integers are resource hogs

Flow Status	Successful - Tue Sep 19 12:43:12 2017
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Standard Edition
Revision Name	counter
Top-level Entity Name	traditionalcounter
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	17 / 56,480 (< 1 %)
Total registers	45
Total pins	34 / 268 (13 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Negative Numbers

- Represented in 2's Complement notation
- Most significant bit (MSB) is the sign bit
  - 1 signifies negative number
  - 0 signifies positive number

$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$

Sign bit

- Range
  - For n bits the range of numbers that can be represented in 2's complement is:
  - (-2^{n-1}) to $(2^{n-1} - 1)$
  - For 8 bits : -128 to 127

Conversion

- Positive number
  - 2's complement number is the true binary equivalent
  - 12_{10} = 01100 *notice that leading 0 is needed
- Negative number
  - Complement each bit – (one's complement)
  - Add 1 to the one's complement
  - Sign bit will always end up a 1

2's Complement Conversion

- Represent -12 in 2's complement notation
  - 12_{10} = 01100
  - 1's complement = 10011
  - Add 1 + 1
  - -12_{10} = 10100

- Represent -93 in 2's complement notation

2's complement conversion

- If signed binary number is positive (how do we know?) do a straight conversion
  - $00111001 = 57_{10}$
- If signed binary number is negative apply the following steps
  - Invert (complement) each bit
  - Add 1
  - Convert to decimal
  - Add negative sign

2's Complement Conversion

- Convert 11100011 to decimal
  - Invert each bit 00011100
  - Add 1 + 1
  - 00011101 = 29
  - Negate 11100011 = -29_{10}
- Convert 11001101 to decimal

Unsigned and signed data types

- VHDL library *numeric_std* defines unsigned and signed data types
  - Both are arrays of std_logic bits
  - What else is an array of std_logic bits?

Signed and unsigned examples

```
--This is a signed multiplier
--*********************************************
Library ieee;
USE ieee.numeric_std.all;
--*********************************************
ENTITY signed_multiplier IS
   PORT (a, b  :IN  SIGNED(7 downto 0);
         y     :OUT SIGNED(15 downto 0));
END signed_multiplier;
--*********************************************
ARCHITECTURE behavior of signed_multiplier IS
BEGIN
   y <= a * b;
END behavior;
```

```
--This is an unsigned multiplier
--*********************************************
Library ieee;
USE ieee.numeric_std.all;
--*********************************************
ENTITY unsigned_multiplier IS
   PORT (a, b  :IN  UNSIGNED(7 downto 0);
         y     :OUT UNSIGNED(15 downto 0));
END unsigned_multiplier;
--*********************************************
ARCHITECTURE behavior of unsigned_multiplier IS
BEGIN
   y <= a * b;
END behavior;
```

Casting

- Industry standard is to use STD_LOGIC_VECTOR at the entity level and cast internal signals to signed and/or unsigned.

```
   --This is an signed multiplier
--*******************************************
Library ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.all;
--*******************************************
ENTITY signed_multiplier IS
   PORT (a, b  :IN  STD_LOGIC_VECTOR(7 downto 0);
         y     :OUT STD_LOGIC_VECTOR(15 downto 0));
END signed_multiplier;
--*******************************************
ARCHITECTURE behavior of signed_multiplier IS
   SIGNAL a_signed, b_signed : SIGNED(7 downto 0);
BEGIN
   a_signed <= SIGNED(a);
   b_signed <= SIGNED(b);
   y <= STD_LOGIC_VECTOR(a_signed * b_signed);
END behavior;
```

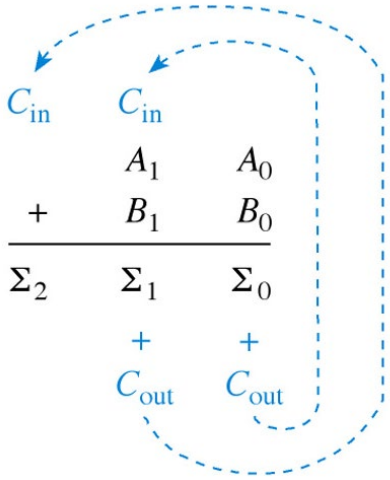
```
--This is an unsigned multiplier
--*******************************************
Library ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.all;
--*******************************************
ENTITY unsigned_multiplier IS
   PORT (a, b  :IN  STD_LOGIC_VECTOR(7 downto 0);
         y     :OUT STD_LOGIC_VECTOR(15 downto 0));
END unsigned_multiplier;
--*******************************************
ARCHITECTURE behavior of unsigned_multiplier IS
   SIGNAL a_unsigned, b_unsigned : UNSIGNED(7 downto 0);
BEGIN
   a_unsigned <= UNSIGNED(a);
   b_unsigned <= UNSIGNED(b);
   y <= STD_LOGIC_VECTOR(a_unsigned * b_unsigned);
END behavior;
```

Arithmetic Operations

- Addition (+)
- Subtraction (-)
- Multiplication(*)
- Division (/)
  - Division is not well supported by all synthesizers

Addition and Subtraction

- The sum needs to be one bit longer than the addends
- Why?



- However, at least one of the operands has to be the same length as the sum

Addition and Subtraction

```
--This is an unsigned adder
--*********************************************
Library ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.all;
--*********************************************
ENTITY unsigned_adder IS
   PORT (a, b  :IN  STD_LOGIC_VECTOR(7 downto 0);
         y     :OUT STD_LOGIC_VECTOR(8 downto 0));
END unsigned_adder;
--*********************************************
ARCHITECTURE behavior of unsigned_adder IS
   SIGNAL a_unsigned, b_unsigned : UNSIGNED(7 downto 0);
BEGIN
   a_unsigned <= UNSIGNED(a);
   b_unsigned <= UNSIGNED(b);
   y <= STD_LOGIC_VECTOR( ('0' & a_unsigned) + ('0' & b_unsigned));
END behavior;
```

Use concatenation to make the operands 1 bit longer

What is the difference and why?

```
--This is an signed adder
--*********************************************
Library ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.all;
--*********************************************
ENTITY signed_adder IS
   PORT (a, b  :IN  STD_LOGIC_VECTOR(7 downto 0);
         y     :OUT STD_LOGIC_VECTOR(8 downto 0));
END signed_adder;
--*********************************************
ARCHITECTURE behavior of signed_adder IS
   SIGNAL a_sign, b_sign : SIGNED(7 downto 0);
BEGIN
   a_sign <= SIGNED(a);
   b_sign <= SIGNED(b);
   y <= STD_LOGIC_VECTOR( (a_sign(7) & a_sign) + (b_sign(7) & b_sign));
END behavior;
```

Multiplication and Division

- The length of the product must be equal to the sum of the lengths of the two numbers being multiplied

```
   --This is an signed multiplier
--*********************************************
Library ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.all;
--*********************************************
ENTITY signed_multiplier IS
   PORT (a, b  :IN  STD_LOGIC_VECTOR(7 downto 0);
         y     :OUT STD_LOGIC_VECTOR(15 downto 0));
END signed_multiplier;
--*********************************************
ARCHITECTURE behavior of signed_multiplier IS
   SIGNAL a_signed, b_signed : SIGNED(7 downto 0);
BEGIN
   a_signed <= SIGNED(a);
   b_signed <= SIGNED(b);
   y <= STD_LOGIC_VECTOR(a_signed * b_signed);
END behavior;
```

- For division, the size of the result must be equal to the size of the numerator.
  - Example is a homework problem ☺

Relational Operations

- Standard logic vectors are evaluated as numbers in relational operations
- Example

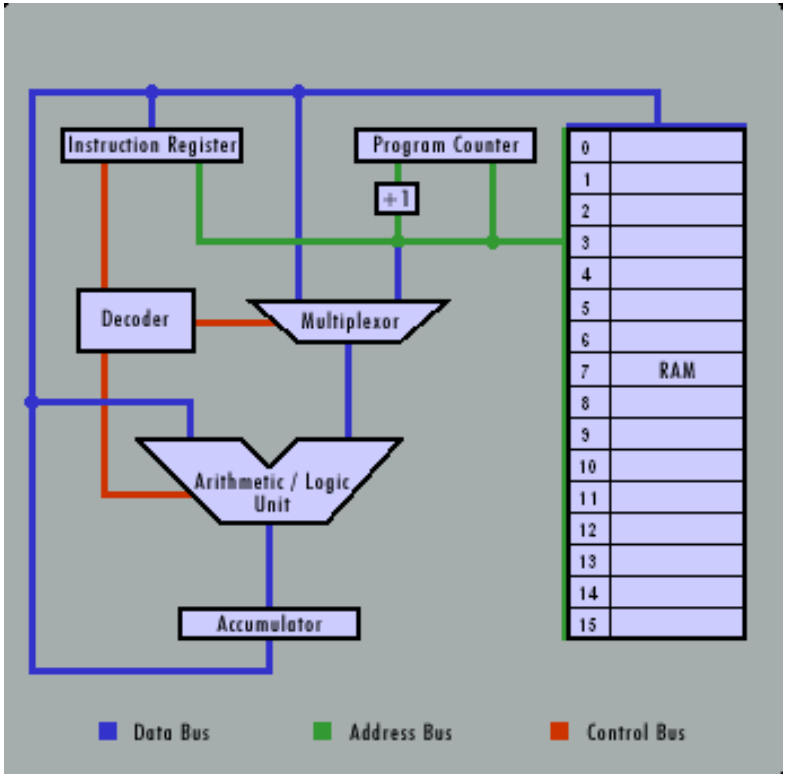
```
A <= “100100”;
B <= “011011”;

IF ( A < B ) THEN
    do this;
ELSE
    do that;
END IF;
```

  - Is it This or That?

ALU's

- Arithmetic Logic Units
  - Circuitry in microprocessor that performs arithmetic and logic operations



What is an ALU?

- ALU is a digital circuit that performs Arithmetic (Add, Sub, . . .) and Logical (AND, OR, NOT) operations.
  - **A and B:** the inputs to the ALU (aka operands)
  - **R:** Output or Result
  - **F:** Code or Instruction from the Control Unit (aka as op-code)
  - **D:** Output status; it indicates cases such as:
    - carry-in
    - carry-out,
    - overflow,
    - division-by-zero