Laboratory 2

(Due date: **005**: February 14th, **006**: February 15th)

OBJECTIVES

- ✓ Implement a large combinational circuit using the Structural Description in VHDL.
- ✓ Implement and simulate floating point and fixed point units in VHDL.

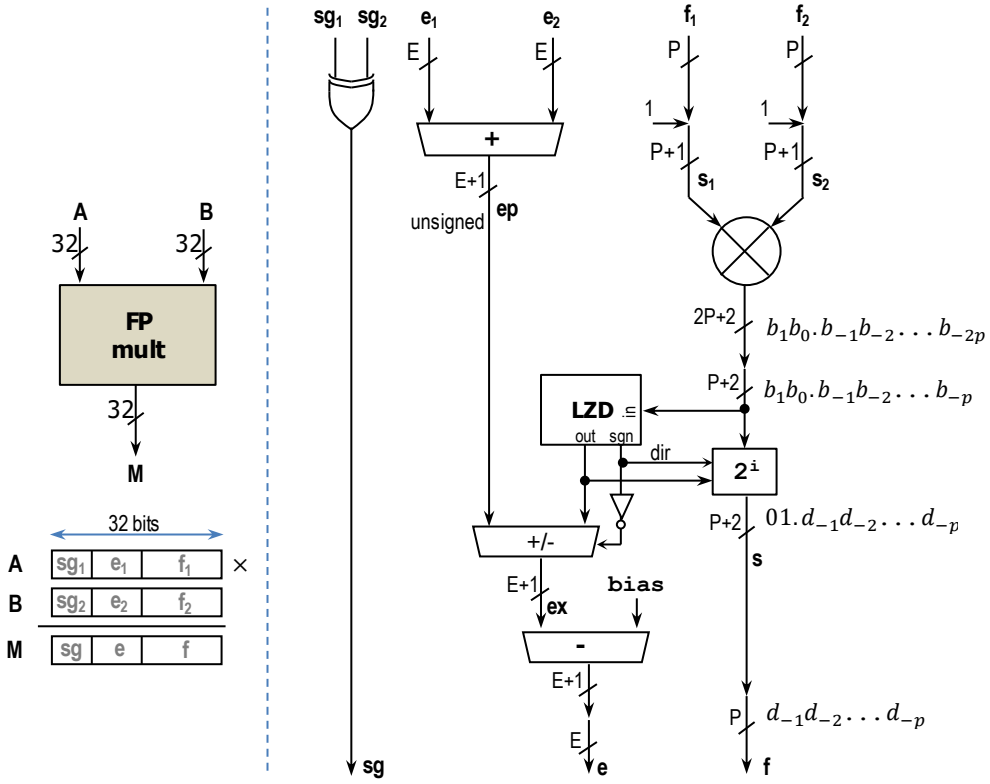
VHDL CODING

- ✓ Refer to the Tutorial: VHDL for FPGAs for parametric code for: adder/subtractor.

FIRST ACTIVITY: FLOATING POINT MULTIPLIER (100/100)

DESIGN PROBLEM

- Implement the following single-precision (E=8, P=23), floating point multiplier. The circuit only works for ordinary numbers generating ordinary numbers (e.g.: the cases $A = 0$ or $B = 0$ are not considered by this circuit). Also, overflow and underflow are not detected by this circuit. The exponents e_1, e_2, e are biased exponents, so they always are positive numbers.



- **Fixed Point Multiplier:** This is an unsigned multiplier.

	Operands		Format (unsigned FX)
Inputs	$s_1 = 1.f_1$	$s_2 = 1.f_2$	[P+1 P]
Output	$s_{12} = b_1b_0.b_{-1}b_{-2} \ldots b_{-2p}$		[2P+2 2P]

- ✓ Truncation: The significand can only have P+2 bits, thus the multiplier output is truncated to P+2 bits (LSBs dropped).
- ✓ Implementation: Use a simple unsigned combinational multiplier:
  Suggestion:

```
use ieee.std_logic_unsigned.all;
...
signal X, Y: std_logic_vector (23 downto 0);
signal Z: std_logic_vector (47 downto 0);
...
Z <= X*Y;
```


Instructor: Daniel Llamocca
TAs: Alex Fillmore, Luke Nuculaj

- **Leading Zero Detector (LZD)**: This circuit outputs an integer number that indicates the amount of shifting required to normalize the result of the multiplication. It is also used to adjust the exponent. This circuit is commonly implemented using a priority encoder. `result` $\in [-1, p]$. The result is provided as sign and magnitude. Use the following code: `myLZD.vhd`.

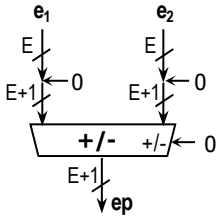
	Operands	Bitwidth
Input	`in`: $b_1 b_0 . b_{-1} \dots b_{-p}$	P+2
Outputs	`sgn`	1
	`out`	$\lceil \log_2 P + 1 \rceil$

  - ✓ The following table details how the expected result is encoded into the signals `out` and `sgn`.

`result`	`out`	`sgn`	Actions
$[0, p]$	$sh \in [0, p]$	0	The barrel shifter needs to shift to the left by sh bits. Exponent adder/subtractor needs to subtract sh from the exponent ep.
-1	$sh = 1$	1	The barrel shifter needs to shift to the right by 1 bit. Exponent adder/subtractor needs to add 1 to the exponent ep.

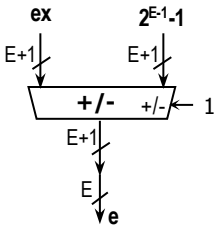
- **Barrel shifter 2^i**: It performs normalization of the final summation. We shift to the left (from 0 to P bits) or to the right (1 bit). Use the VHDL code `mybarrelshift_gen.vhd` with `SHIFTTYPE="LOGICAL"` (unsigned input), `dir=sgn(LZD)`.

- **Exponents adder**: We need to add the biased exponents: $ep = e_1 + e_2$. This is an unsigned addition that results in at most E+1 bits. Thus, we can use a simple 2C adder/subtractor, where we zero-extend the input operands to E+1 bits. The output ep is an unsigned number with E+1 bits.



- **Exponent adder/subtractor**: The input operands are unsigned with E+1 bits: ep and `out` (from LZD, it needs to be zero-extended to E+1 bits). It can be shown that the result (ex) cannot be negative.
  - ✓ Implementation: you can use a 2C adder/subtractor. Strictly speaking, you need to zero-extend the operands to E+2 bits. But since the result is always positive, you can use the 2C adder/subtractor with E+1 bits, where the output ex is an unsigned number with E+1 bits.

- **Bias subtractor**: The input operands are unsigned with E+1 bits: ex and `bias` ($2^{E-1} - 1$).
  - ✓ Implementation: As in the previous 'exponent adder/subtractor', you can use a 2C adder/subtractor. Strictly speaking, you need to zero-extend the operands to E+2 bits. But since the output is always positive, you can use E+1 bits in the 2C adder/subtractor, where the output is an unsigned number with E+1 bits.
  - ✓ Since we are subtracting the $bias$, it can be shown that the unsigned result only needs a maximum of E bits. Thus, by dropping the MSB, we get the final exponent e.



PROCEDURE

- **Vivado: Complete the following steps:**

  - ✓ Create a new Vivado Project. Select the corresponding Artix-7 FPGA (e.g.: the XC7A50T-1CSG324 for the Nexys A7-50T).

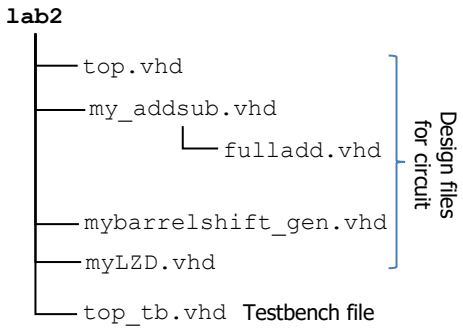
  - ✓ Write the VHDL code for the 32-bit floating point multiplier. Synthesize your circuit to clear syntax errors.
    You MUST use the Structural Description: Create a separate `.vhd` file for the components (adder/subtractor, Barrel Shifter, LZD) and interconnect them all in a top file.

  - ✓ Write the VHDL testbench to test the following cases:
    ECE4710F × CAFE0080 = 7862A89F
    DECADE80 × 4710FACE = E665C7D1
    BA5EBA11 × F007BA11 = 6AEC2C01
    0FACADE0 × 50F7BA11 = 21271944
    EA514710 × BEA7AB1E = 6989113E

  - ✓ Perform Functional Simulation and Timing Simulation of your design. **Demonstrate this to your TA**.
    Note that when testing, it might be very useful to represent the inputs and output in single floating point precision. Or we might also want to represent the intermediate signals as either integer numbers or fixed-point numbers. You can use the Radix → Real Settings in Vivado simulator window to do so.

SUBMISSION

- Submit to Moodle (an assignment will be created):
  - ✓ This lab sheet (as a .pdf) completed and signed off by the TA (or instructor).
  - ✓ (As a .zip file) All the generated files: VHDL design code, VHDL testbench. **DO NOT submit the whole Vivado Project** (points will be deducted otherwise).
    - Your .zip file should only include one folder. Do not include subdirectories.
    - It is strongly recommended that all your design files, testbench, and constraints file be located in a single directory. This will allow for a smooth experience with Vivado.
    - You should only submit your source files AFTER you have demoed your work. Submission of work files without demoing will be assigned NO CREDIT.



TA signature: _____ **Date:** _____

Instructor: Daniel Llamocca
TAs: Alex Fillmore, Luke Nuculaj