Laboratory 2

(Due date: **005**: February 17th, **006**: February 18th, **007**/**008**: February 19th)

OBJECTIVES

- ✓ Implement a large combinational circuit using the Structural Description in VHDL.
- ✓ Introduce floating point and fixed point representations for VHDL implementation and Vivado simulation.

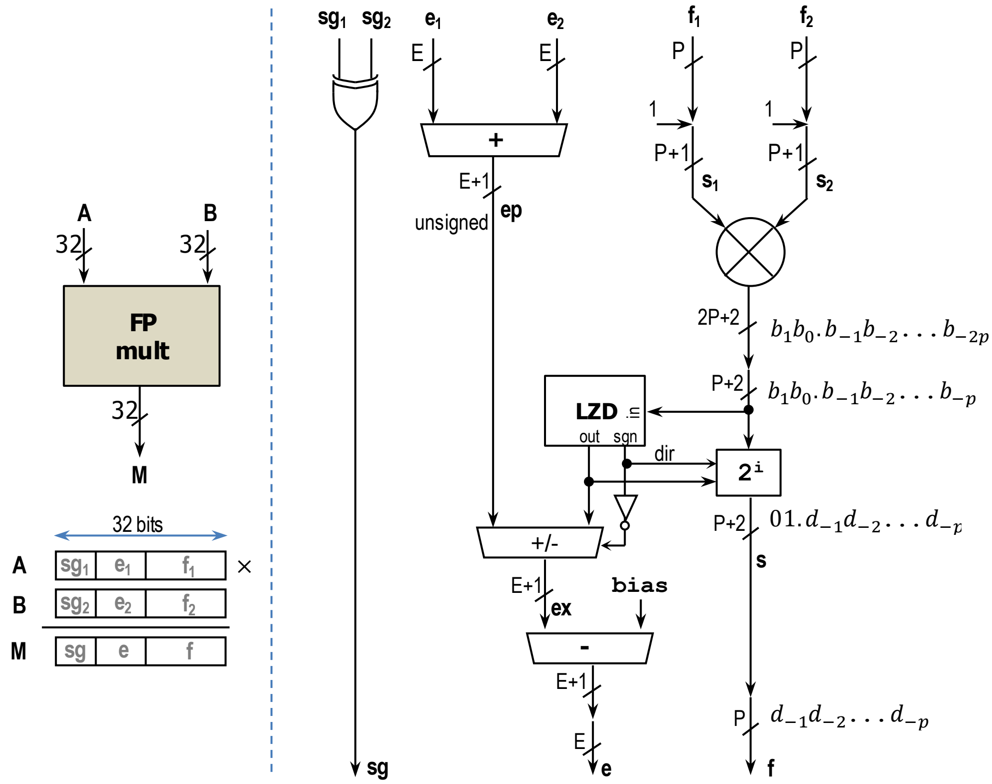
VHDL CODING

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

ACTIVITIES

FIRST ACTIVITY: FLOATING POINT MULTIPLIER (100/100)

- 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 in the circuit 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	$s_{12} = b_1b_0.b_{-1}b_{-2}...b_{-2p}$		[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: Andrew DeJonge, Nathan Kelley

- **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`.

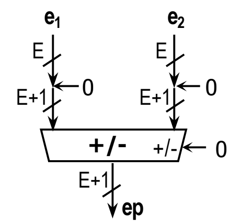
	Operands	Bitwidth
Input	`in`: $b_1 b_0 . b_{-1} \dots b_{-p}$	P+2
Outputs	`sgn`	1
	`out`	$\lceil \log_2 P + 1 \rceil$

  ✓ 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.

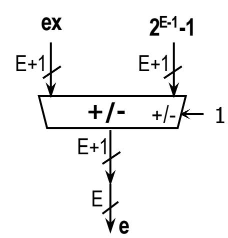
- **Barrel shifter 2^i**: It performs normalization of the final summation. We shift to the left (from 0 to P bits) or to the right (1 bit). Use the VHDL code `mybarrelshift_gen.vhd` with `SHIFTTYPE="LOGICAL"` (unsigned input), `dir=sgn(LZD)`.

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



- **VIVADO DESIGN FLOW FOR FPGAs – NEXYS A7-50T**
  ✓ 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 adder subtractor. Utilize the Structural Description: create a separate 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:
  `7A09D300 × 0BEEF000 = 4680A35F`
  `7A09C000 × 8BEE0000 = C6801080`
  `01800000 × FAB80000 = BCB80000`
  `0A800000 × FAB80000 = C5B80000`
  `80C00000 × FAD00000 = 3C1C0000`

  ✓ 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 not only as integer numbers but also as fixed point numbers. You can use the `Radix` → `Real Settings` in Vivado simulator window to do so.

- Submit (as a .zip file) all the generated files: VHDL code files and VHDL testbench to Moodle (an assignment will be created). DO NOT submit the whole Vivado Project.

TA signature: ______________________________ **Date:** ________________________