Lab 11 - Trigonometric Functions on FPGAs

In this lab, we'll implement a module that returns the sine, cosine and tangent of a input variable X. These are the interface port of the module.

Port	Direction	Type	Width
CLK	IN	std_logic	1
X	IN	unsigned	8
SEL	IN	std_logic_vector	2
RESULT	OUT	signed	9

A testbench is provided to test your design.

Depending on the value of SEL, the module returns one of three implemented functions, according to the following truth table.

SEL	RESULT
00	0
01	COS(X)
10	SIN(X)
11	TAN(X)

Exercise

1. Deal with floating point numbers.

The first step is to deal with the integer representation of X and RESULT. For that, we have to introduce the concept of range and multipliers.

In our case X is an unsigned variable that can take values between 0 and 2^8-1. Since X, should represent an angle between 0 to 2п, we can calculate the multiplication factor or multiplier in this way:

$X_{mult} = (X_{range})/(X_{max}^{float} - X_{min}^{float}) = (2^8 - 1)/(2п-0) = 40.584510488$

It is however convenient to choose a multiplier which is a power of 2, to ease the calculations on FPGAs. This will impact the actual allowed range of `X` and also its granularity. In our case, the closest multiplier that is a power of 2, that allows a full angle range is 32.

That means that `X` values will float between 0 and (2^8-1/32)=7.96875.

The digitised angles are then represented by the equation

$X = int(X_{float} * 32)$

For the first exercise, calculate a reasonable multiplier for the sine, cosine and tangent functions. Use multipliers that are power of 2.

2. Define the ROMs

Go to the `~/labs/lab11/` folder, and open the `src/trigonometric.vhd` file with a text editor, and implement the ROM following the comments in the file and the following suggestions.

```
kate src/trigonometric.vhd
```

With the calculated multipliers, you can now create the ROMs that will store the values for the sine and cosine. Each address in the ROM corresponds to a value of `X`, and its content is the corresponding sine or cosine.

A python script is available in `scripts/generate_vhdl_array.py`, which can be used to generate the initial content of the ROM. This is the usage of the script:

```
$> python3 scripts/generate_vhdl_array.py -h
usage: generate_vhdl_array.py [-h] [--type {sin,cos,tan}] input_mult
output_mult depth
```

Once you defined the ROMs, code the synchronous process that, depending on the value of `SEL`, reads the corresponding ROM and assign the read value to the `RESULT` port.

3. Run the simulation

To check your code run the simulation script.

```
./run_sim.sh
```

Does your simulation pass? If it fails, check the log, and understand what the problem is.

The testbench is assuming a multiplier of 64 for all three trigonometric functions. If you chose different multipliers, open the `sim/tb_trigo.vhd` file, and change lines 34-36 accordingly. Re-run the simulation and verify that everything went well this time.

4. Initialise the memory from a file

The python script should also have generated three `.dat` files containing the ROM contents.

```
data/sin.dat
data/cos.dat
data/tan.dat
```

Define an `impure function` in your module, which initialises the ROM from a file. It should get as argument the filename as string.

N.B. The `read` procedure returns a `std_logic_vector`: https://portal.cs.umbc.edu/help/VHDL/packages/std_logic_textio.vhd

Test your code again, by running the simulation