Classiq's Generalized Arithmetic Challenge

In classical computing, nonlinear functions such as trigonometric functions, hyperbolic functions, and fractional powers are the bread and butter of algorithm development. In turn, these generalized mathematical operations can be described by basic arithmetic ones, e.g., addition and multiplication, through polynomial approximations.

Generalized arithmetic functions are at the heart of many fault-tolerant quantum algorithms. For example, the HHL algorithm contains an eigenvalue inversion block that can be implemented with an arcsin(1/x) function. Another example is the implementation of some non-linear payoff-function in quantum applications for option-pricing.

Specifically, the tanh(x) function might be used in statistical mechanics applications like Gibbs state preparation or as an activation function in quantum machine learning. As the field progresses, it is highly probable that more and more use cases of such functions will be found.

Even with today's noisy-intermediate-scale-quantum (NISQ) machines (or utility era), implementation of generalized arithmetics can be used for benchmarking devices or as a tool for near-term quantum algorithms.

Designing quantum generalized arithmetic algorithms is not an easy task: the reversible nature of quantum computing and the non-cloning theorem force the usage of out-of-place operations and additional auxiliary registers, when connecting basic arithmetic operations to construct a polynomial approximation.

Luckily, the Classiq platform enables you to focus only on the high-level implementation and algorithmic considerations, and it automatically optimizes your algorithm with all the complex low-level considerations that need to be made.

Challenge Definition

The challenge deals with polynomial, or piecewise polynomial approximations of the tanh(x) function. The task is to find the most accurate approximation of tanh(x) in the domain $f(x)$: $[0, 1) \rightarrow [0, 1)$**.**

You will need to create a quantum model which implements the following operation

$$|x\rangle_n \rightarrow |x\rangle_n |f(x)\rangle_m$$

The precision of your input states (i.e. the number of fraction digits of the numbers the input state represent) is determined by the number of qubits n. In your algorithm, it must be a parameter, such that the following function (with this exact declaration) should appear in your code:

```
@QFunc
def compute_tanh(precision: QParam[int],    # The precision of x
                 x: QNum,                   # Quantum variable that represents numbers
                 tanh_x: Output[QNum]       # The output of the function
                 ):

   pass #TODO delete pass and write your code here
```

The precision of the output m can be anything you want and might depend on the precision of the input n.

The high level goal is to design the most accurate approximation for a quantum computer with 100 qubits and a circuit depth of 30,000 (assuming a logical quantum computer, errors and error mitigation / correction should not be taken into account here). As one would do in a general research process, before attempting the large task directly, a small example that represents the large target should be devised and tested. The main idea is that you will generate 1 quantum model, your algorithm, that will be adapted with the Classiq synthesis engine (compiler) one time for the small example, and one time for the large example. Therefore, your code should be parametric as described in detail in the accompanied Jupyter notebook.

There are 2 stages for the task:

Stage 1

The goal is to design the most accurate approximation with the lowest maximal distance between your solution and the ground truth (see the `evaluate_score_stage_1(results, user_input_precision)` function in the notebook). Your solution must hold the following criteria:

- **Maximum** circuit width of 30 qubits.
- **Maximum** circuit depth of 3,000.

The checking mechanism for this stage is available in the notebook for your convenience. **The top 10% of participants with the lowest maximal distance will pass to stage 2.**

Stage 2

The goal is to design the most efficient algorithm that results in the most shallow circuit with the highest precision possible. Your solution can use up to 100 qubits, and it must hold the following criteria:

- **Maximum** circuit depth of 30,000.
- **Maximum** circuit width of 100 qubits.

The solution will be judged according to the maximal precision n and the minimal depth of the circuit. The elegancy and originality of the solution might also be taken into account.

Note: Both stages will be checked with the same *compute_tanh(PRECISION, x, y)* function, so it needs to be parametrized. The first stage is meant to validate your algorithm with something that can be simulated nowadays, whilst the second stage is the true goal.

What You Need To Submit

You need to fill this Google Form and to upload to it (guidance inside) the following files:

1. Your quantum model as a '.qmod' file
2. Your quantum program for stage 1 as a '.qprog' file
3. Your quantum program for stage 2 as a '.qprog file

The quantum model is the file that captures your algorithm, whilst the quantum program contains the compiled quantum circuit (as well as the execution preferences of the quantum circuit). In the accompanied notebook you should check the 'Saving The Files' section with detailed instructions.

Support & Getting Started

- Go over the accompanied Jupyter notebook.
- The Classiq Platform can be accessed at **platform.classiq.io** and the documentation can be found here.
- Our vibrant Slack community is here to help you with anything you need during the challenge and in general!

Good Luck!