CLASSIQ

WHAT IS QUANTUM ALGORITHM DESIGN?

Application note | August 2021

Introduction: What is Quantum Algorithm Design?

Quantum Algorithm Design (QAD) is the quantum version of computer-aided design (CAD). With QAD, quantum software engineers and scientists innovate and produce much faster than ever before. Like in traditional computer-aided design, QAD users achieve extraordinary results by letting computers handle the things that computers are good at, freeing users to think, invent, and innovate.

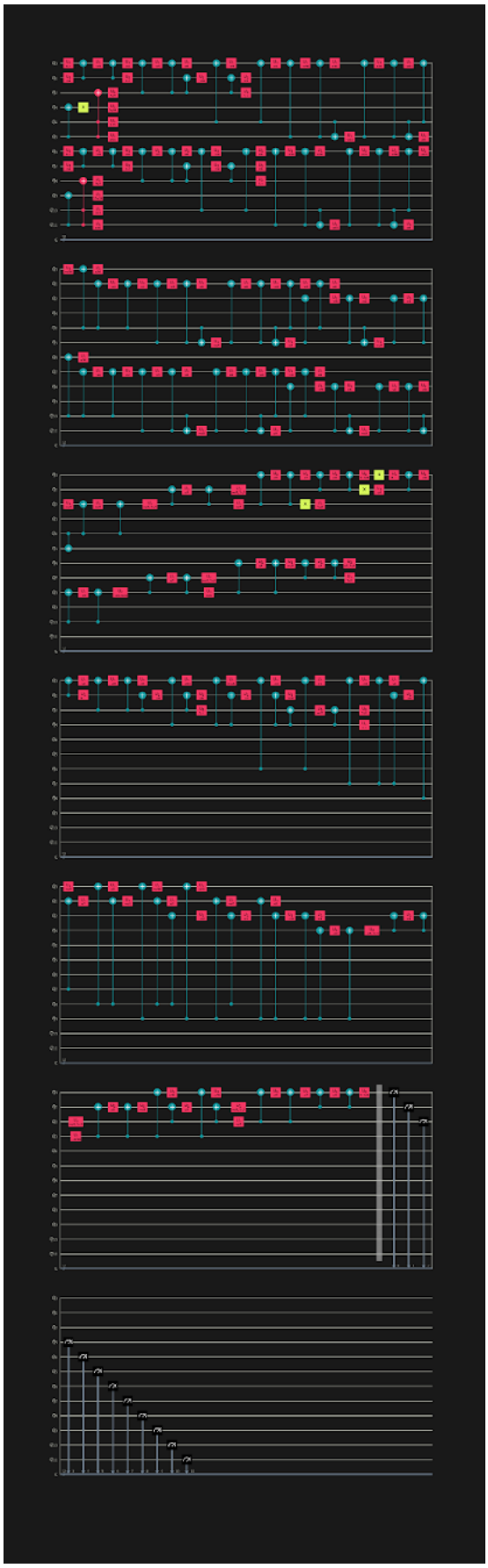
QAD aims to solve the same problem that was solved years ago by electronic CAD: as circuits become increasingly larger and more complex, it becomes impossible to design them by hand. While an electronic engineer can undoubtedly put together a working circuit with 20-30 logical gates, today's newer electronic ICs have millions and sometimes billions of gates. Creating the Netlist for these chips by hand is simply impossible. However, providing a high-level functional model to a computer and asking the computer to convert this into a working circuit is most certainly possible, hence the popularity of CAD platforms.

What does Classiq do?

Classiq's Quantum Algorithm Design platform automatically synthesizes complete quantum circuits from high-level functional models. What does this mean? It means that within seconds you can get from a high-level functional description like this quantum arithmetic code fragment:

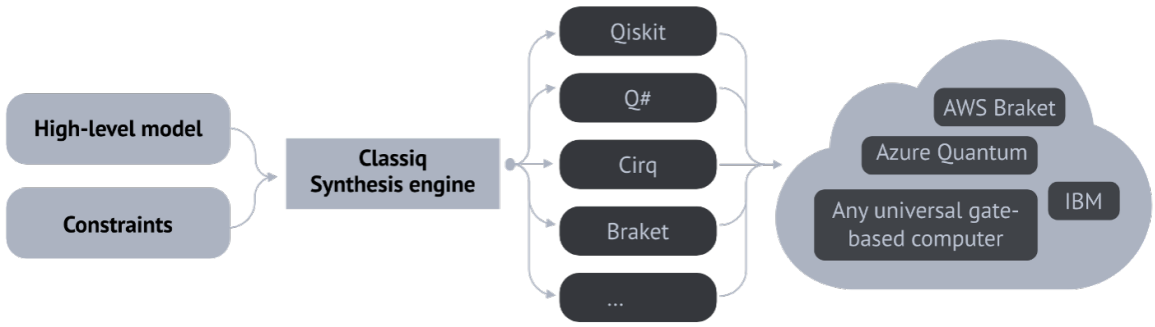
```
{
   "qubits_count": 12,
   "min_depth": 1,
   "max_depth": 10,
   "segments": [
   {
      "function": "ArithmeticExpression",
      "qubits_count": 12,
      "symbolic_statement": "(a+b) * (a-b)",
      "register_sizes": {"a": 2, "b": 2},
      "optimization_criteria": "depth",
      "add_as_single_gate": false
   }
]
}
```

to a working quantum circuit that implements this high-level functionality while meeting the designer-specified constraints (more on constraints later). Such a circuit is shown on the right.



Where does Classiq fit in the quantum software stack?

The Classiq engine ingests a high-level functional model of the desired quantum circuit and a constraints file. It can output code in various quantum languages, including Qiskit, Q#, Cirq, and more. Furthermore, it includes pre-configured integrations into most major quantum cloud providers, including IBM, Amazon Braket, and Azure Quantum.



The output of the Classiq engine is an agnostic quantum circuit, described in any gate-level programming language. Compilers and transpilers ingest that output and adapt it to the particular hardware of choice.

The ability to output code in various formats and be compatible with a variety of cloud providers means that it is very easy to port code from one hardware target to another. These days, when the industry is still in development, some companies are hesitant to commit to one particular hardware architecture. Thus, the ability to deploy quantum circuits across a wide range of hardware back-ends is essential.

How is QAD different from a compiler?

The Classiq Quantum Algorithm Design platform does not replace quantum compilers or transpilers. Compilers ingest the output of the Classiq platform and have an important role to perform, adapting the code to the particular hardware-specific connectivity and available gates.

Where QAD adds significant value is in system-level optimization and in satisfying the constraints dictated by the designer.

A compiler can perform specific optimizations because it understands the connectivity and properties of the target hardware. A compiler can also perform local optimizations such as eliminating two back-to-back Hadamard gates.

QAD, in contrasts, provides system-level optimizations. For instance, when creating a circuit for quantum arithmetic, QAD can preserve intermediate values if they are used downstream or recover the qubits that hold them and use them for some other purpose. QAD can do that because it understands the intent of the algorithm designer. A compiler that looks at a concrete gate-level quantum circuit (in QASM or another format) cannot.

QAD platforms analyze thousands upon thousands of options to come up with the optimal solution. This level of analysis and optimization simply does not exist in compilers.

What are constraints in the QAD context?

Just like different people consider different constraints and wishes when they buy a home, different quantum designers have different constraints that they want to impose on the output circuit. These could be driven by hardware constraints, personal preference, or a host of other reasons.

Here are some of the constraints that the Classiq platform can handle:

- The width and depth of the circuit. How many qubits can you use? How deep can the circuit be before errors creep in? A designer might want, for instance, to add qubits (increase the width) as a way of reducing the depth.
- The usage of particular types of gates or preferred gate sets. Suggestions from the target hardware vendor might drive this gate set.
- The desired accuracy. As can be seen later in this document, functional blocks such as state preparation can be built in a variety of ways depending on the desired accuracy.
- The connectivity or particular qubits, allowing to minimize the use of swap gates.

One of the unique capabilities of the Classiq platform is that it analyzes many thousands of options to find the best one that satisfies these constraints. The designer can change the constraints and regenerate the circuit to explore various options. Doing this by hand might take days, but with Classiq, it takes seconds.

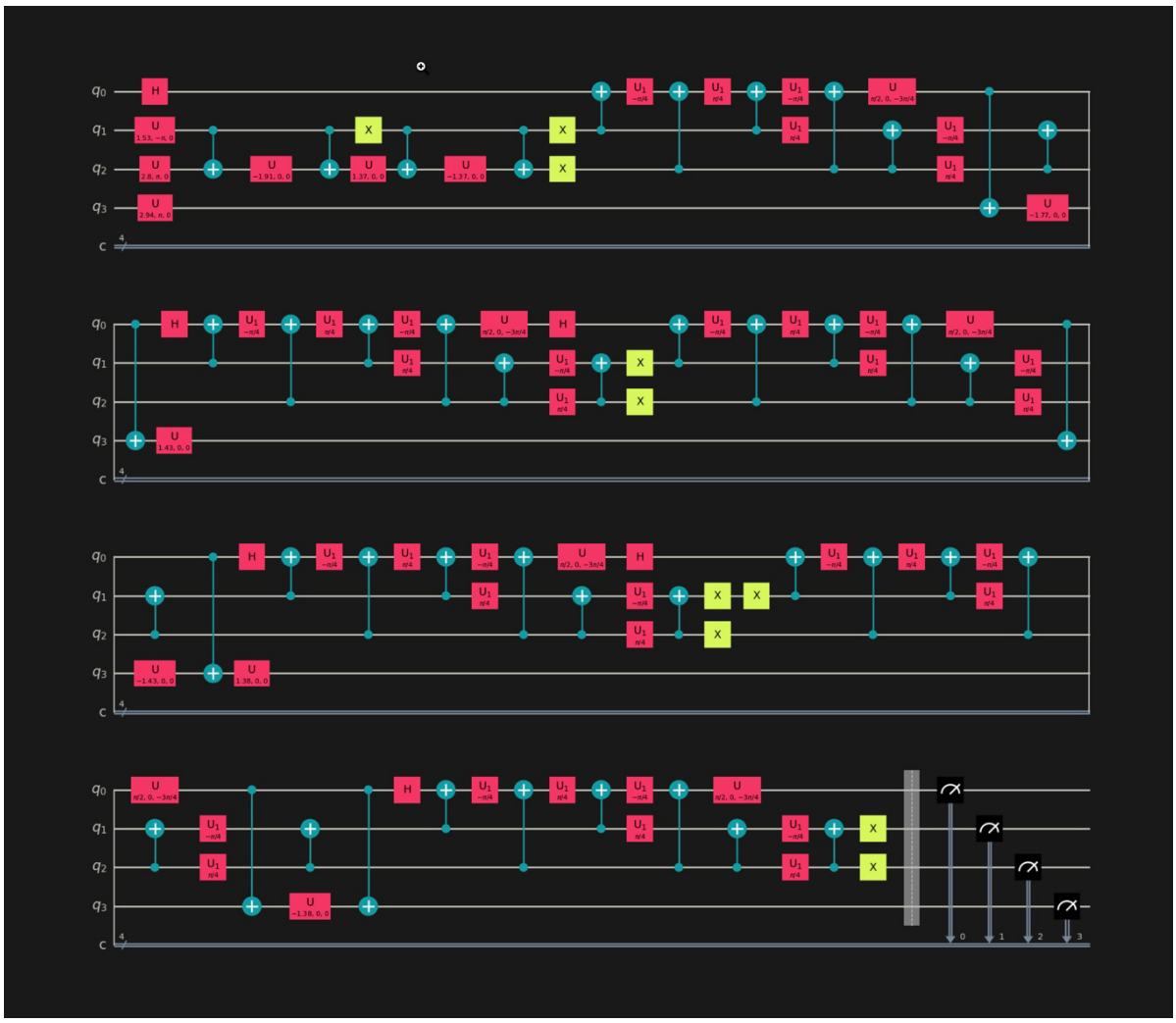
For instance, the following state preparation code (loading probability mass functions):

load-probs.clsq

examples > load-probs.clsq > ...

```
{
    "qubits_count": 4,
    "max_depth": 100,
    "segments": [{
        "function": "StatePreparation",
        "function_params": {
            "probabilities": {"pmf":[0.05, 0.11, 0.13, 0.23, 0.27, 0.12, 0.03, 0.06]},
            "error_metric": {"KL": {"upper_bound": 0.001}}
        }
    }]
}
```

Generates the following circuit:



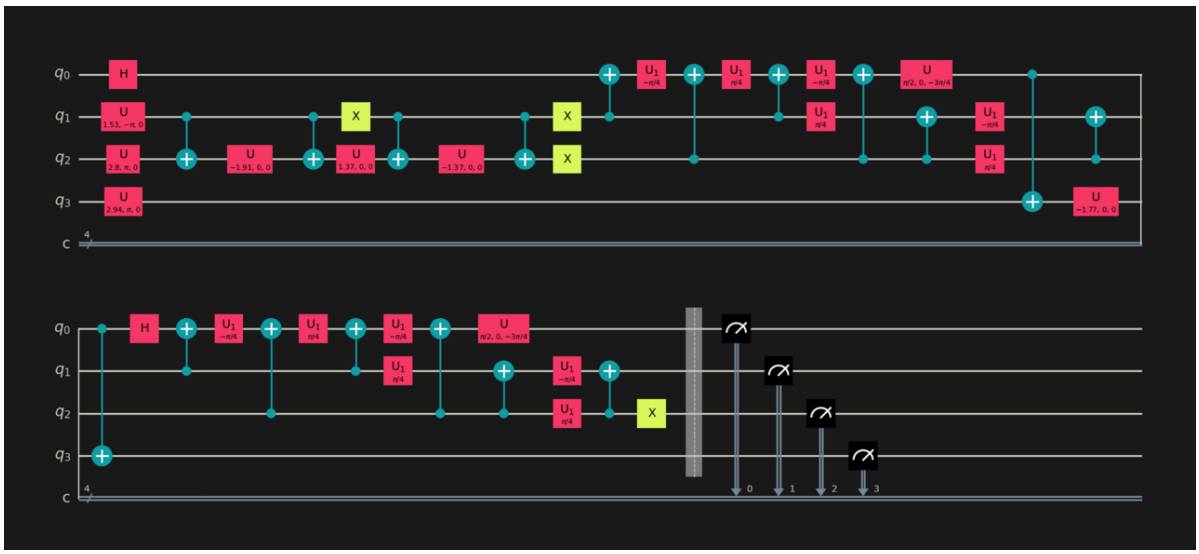
But since this circuit might be too deep. So the designer might try to change the accuracy of the loaded states from 0.01 to 0.05:

load-probs.clsq

examples > {} load-probs.clsq > [ ] segments > {} 0 > {} function_params > {} error_metric > {} KL > # upper_bound

```
{
    "qubits_count": 4,
    "max_depth": 100,
    "segments": [{
        "function": "StatePreparation",
        "function_params": {
            "probabilities": {"pmf":[0.05, 0.11, 0.13, 0.23, 0.27, 0.12, 0.03, 0.06]},
            "error_metric": {"KL": {"upper_bound": 0.05}}
        }
    }]
}
```

Resulting in a simpler circuit:



This demonstrates the division of labor that is so important in QAD: the designer defines the high-level functional model and the constraints and then lets the computer analyze thousands of options to find a circuit that implements this while meeting the constraints.

Can the constraints always be met?

Of course not. It may be that the platform cannot find a solution in a reasonable amount of time. This can happen when the platform indicates that the constraints are unsatisfactory, meaning that the engine proved mathematically that the requirements couldn't be met. Perhaps the number of qubits or depth of the circuit is too small, or other constraints make it impossible. In other cases, the synthesis may just take too long, and the user could elect to relax some of the constraints and shorten the circuit generation time.

What are the advantages of Quantum Algorithm Design?

Quantum algorithm design lets the designer focus on the algorithm's functionality instead of on the low-level implementation. This translates to substantial advantages both today and tomorrow:

"Today," when the quantum computers have a few or at most dozens of qubits, this approach provides dramatic timesaving. It also allows estimating the resources required to run a particular algorithm before spending too much time building it. It may be that some algorithms are just too complex for today's machines, and a quick way to determine this is beneficial.

"Tomorrow," when there will be hundreds or thousands of qubits, we believe that this approach will make the impossible possible. The complexity of these machines will be too much for even highly skilled quantum information scientists. The ability to generate sophisticated algorithms from high-level functional models will be paramount.

Both today and tomorrow, there is a significant advantage to be gained by making quantum more accessible. Just like a Web designer does not need to understand how a CMOS gate works, a quantum software engineer should not need to understand the intricacies of the hardware. By focusing on the functional requirements, teams can integrate experts from other fields. For instance, a financial option pricing expert might join a team using quantum computers for sophisticated pricing models. Similarly, supply-chain experts or chemists can join their company's quantum teams.

If QAD is an abstraction layer, are we losing optimization capabilities?

Not really. It is true that if you are an expert in “to the metal” code, you could theoretically squeeze the last bit of optimization from your software. Still, QAD provides you with other critically important advantages, as well as optimizations that couldn’t be reached otherwise. The synthesis engine also examines many possible solutions - more than a person could realistically examine - and chooses the optimal circuit.

One advantage is the ability to move your code between various hardware providers quickly. It is unclear which platforms will win, and organizations seek to mitigate risks by writing portable code.

Another advantage is that programming “to the metal” quickly becomes unfeasible with the growing complexity of quantum computers. Last, high-level functional code is much easier to debug and maintain than the equivalent of ‘quantum assembly language’

Don’t some existing tools already provide building blocks?

Existing development tools indeed provide some templates, for instance, for VQE. However, customizing those templates requires a lot of work. Search algorithms, for example, require an oracle function, and creating such functions is very easy with QAD yet impossible with any other current methods. A Monte Carlo option pricing circuit, for instance, might require a sophisticated payoff function which will be very difficult to create, debug, and maintain with standard development tools. But such a payoff function will be much easier to create with QAD.

Additionally, if one wishes to create an entirely new algorithm, QAD will make this process much easier.

August 2021

How do I get started with Quantum Algorithm Design?

Get started at www.classiq.io, where you will find technical documents, demonstration videos, and information on how you can contact us.

CLASSIQ

REVOLUTIONIZING THE DEVELOPMENT OF QUANTUM SOFTWARE

In this note, you learned about the need for Quantum Algorithm Design platforms and saw some examples of their use. We hope you had a chance to appreciate how much a QAD platform makes it easier to design sophisticated quantum algorithms.

REQUEST A DEMO TODAY

hello@classiq.io
www.classiq.io