

CREATE, ANALYZE AND OPTIMIZE A SIMPLE QUANTUM CIRCUIT

Application note | June 2021



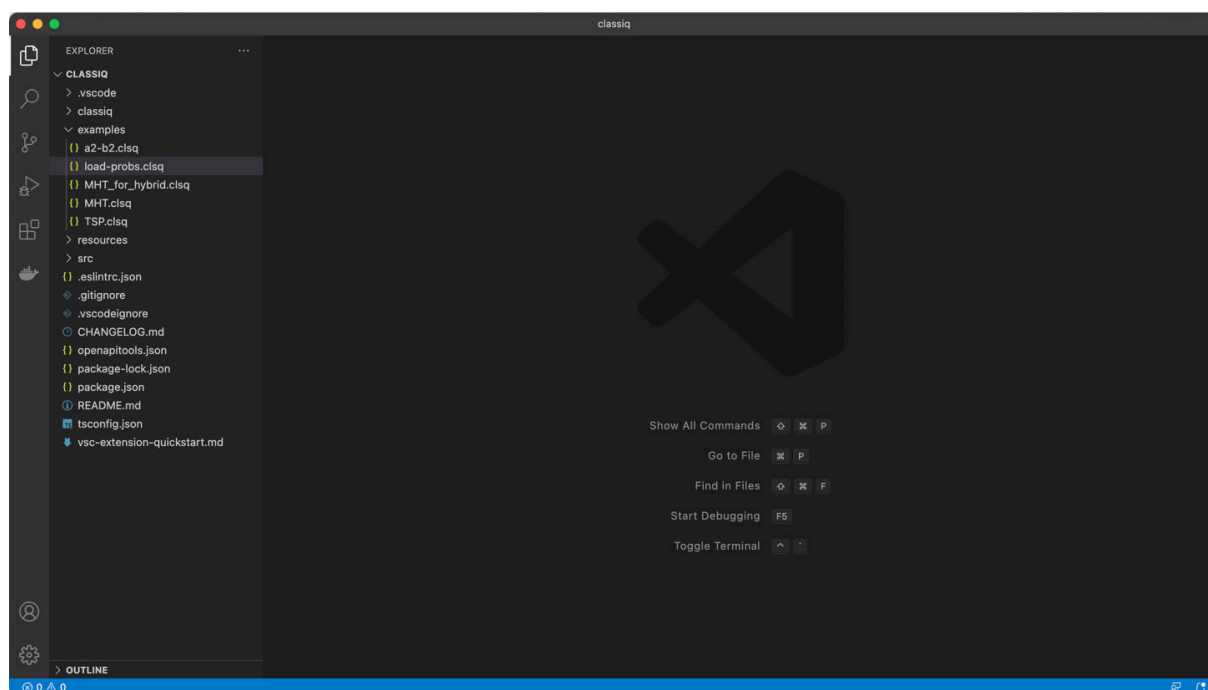
Introduction

State preparation - the process of loading a known probability mass function or setting a quantum system to a specific known state - is a critical part of quantum algorithm development. It is also essential in hybrid classical/quantum algorithms that perform some quantum calculation, which needs to start in a known state, measure the state, perform a classical computation, and repeat this process as necessary, using different starting states for each iteration.

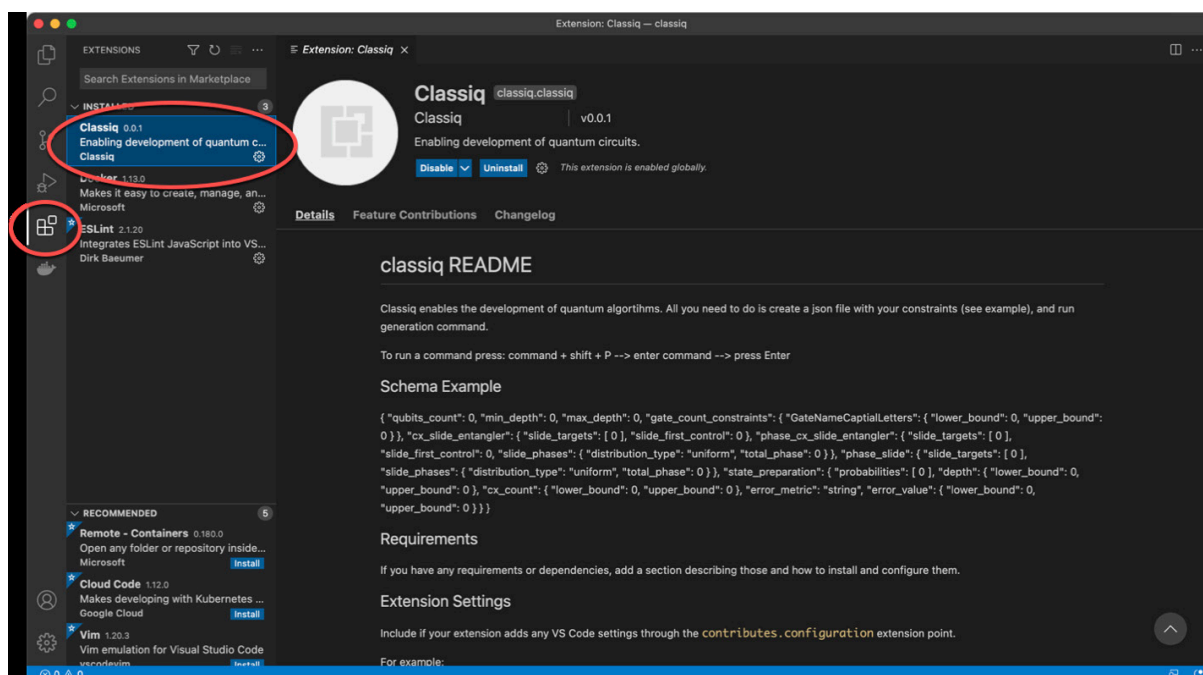
This tutorial will show how to build, analyze, and optimize a simple state preparation circuit using the Classiq Quantum Algorithm Design platform. Perform the following steps:

Open VSCode

VSCode is the environment in which you will create, synthesize and analyze quantum models.

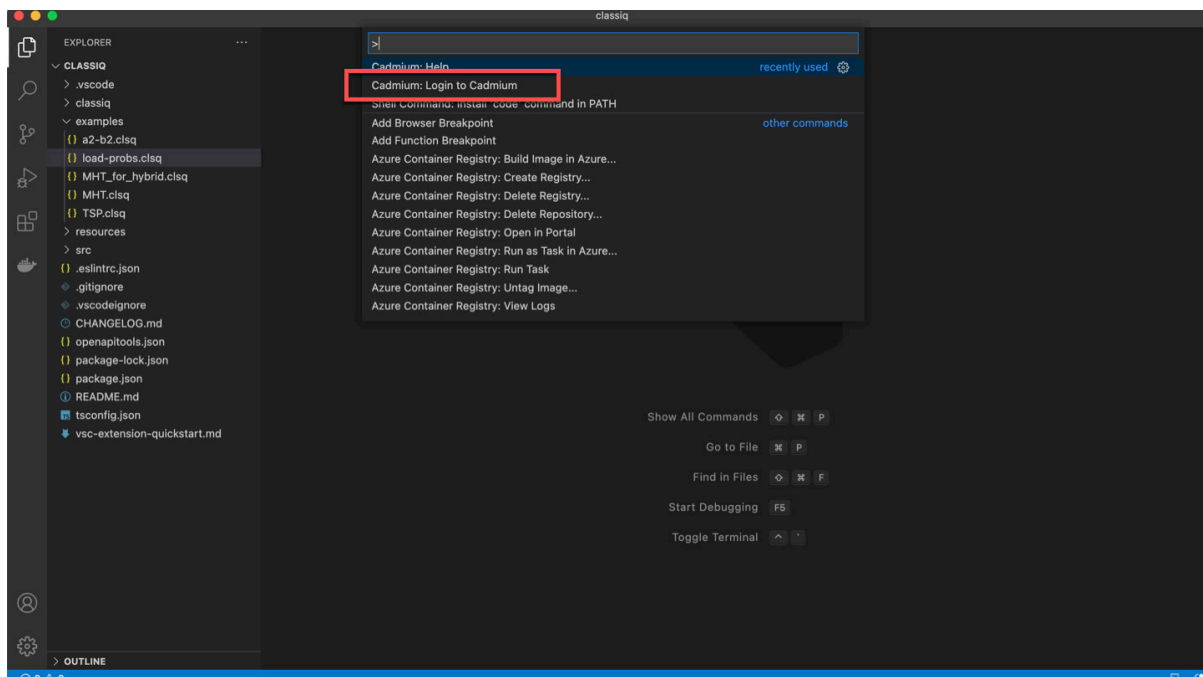


If this is your first time using Classiq, find the Classiq extension and install it:



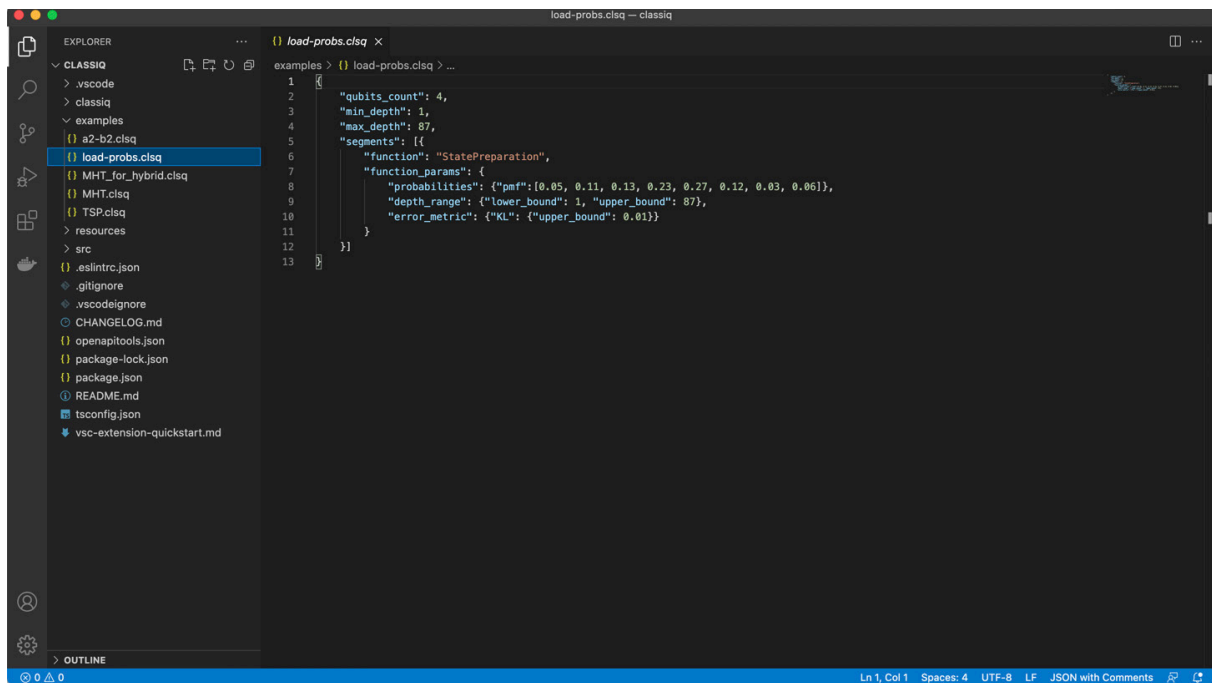
Login

Press `Ctrl+Shift+P` on PC or on `⌘⇧P` Mac to show the command palette and choose "Cadmium: login to cadmium". This will take you to a Web page in which you will enter your login credentials.



Open the example file

Open “load-probs.cldq” file:



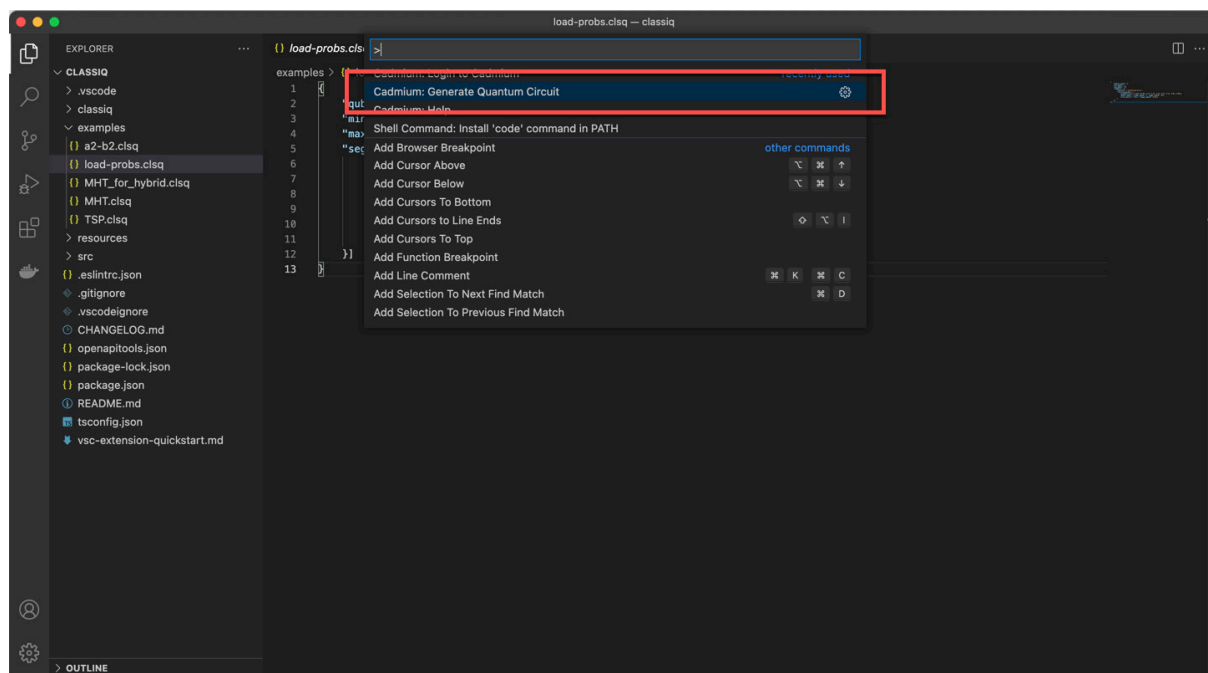
In case you don't have it in your environments, here is the content of this file:

```
{
  "qubits_count": 4,
  "min_depth": 1,
  "max_depth": 87,
  "segments": [{
    "function": "StatePreparation",
    "function_params": {
      "probabilities": {"pmf": [0.05, 0.11, 0.13, 0.23, 0.27, 0.12,
0.03, 0.06]},
      "depth_range": {"lower_bound": 1, "upper_bound": 87},
      "error_metric": {"KL": {"upper_bound": 0.01}}
    }
  ]
}
```

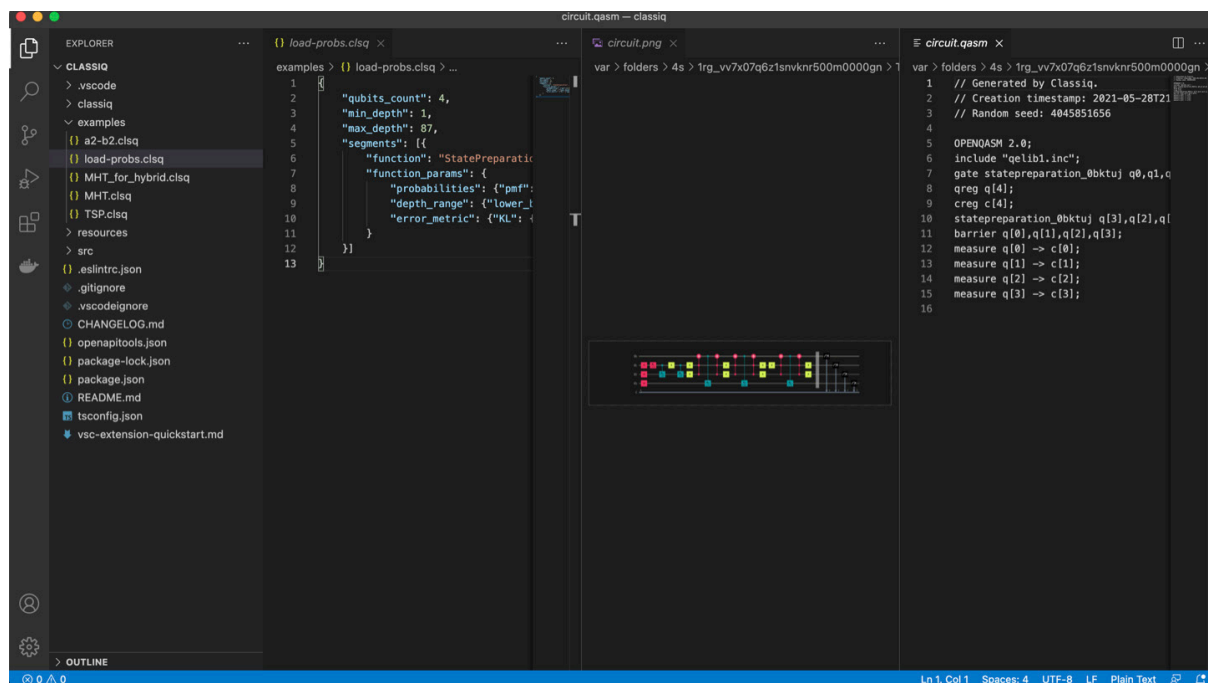
The desired probability mass function (“pmf:” statement) has eight elements, so it is aiming to prepare the state of three qubits (since $2^3 = 8$)

Generate the circuit

From the command palette, select “Cadmium: generate quantum circuit”



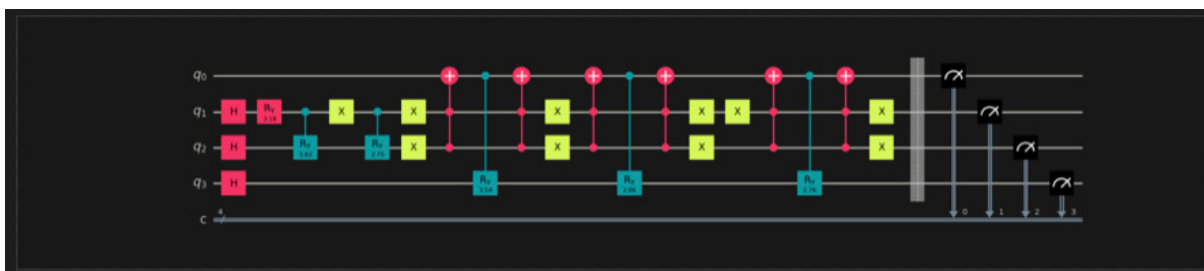
You will be prompted to enter a name for the output, such as test. After a few seconds, you will see a screen like this:



This view has three main parts:

Source code: this is the left panel, and it shows the source code of the model that we loaded

Quantum circuit: in this middle panel, we see the quantum circuit generated by Classiq. It fulfills the required behavior of the model and meets the desired constraints. In this case, the circuit looks like this:



As you can see, this circuit uses four qubits which is what we defined in the model.

```
qubits_count": 4,
```

Qasm code: the panel on the right shows Qasm code that fulfills this circuit:

```
// Generated by Classiq.
// Creation timestamp: 2021-05-28T21:05:45.641814+00:00
// Random seed: 4045851656

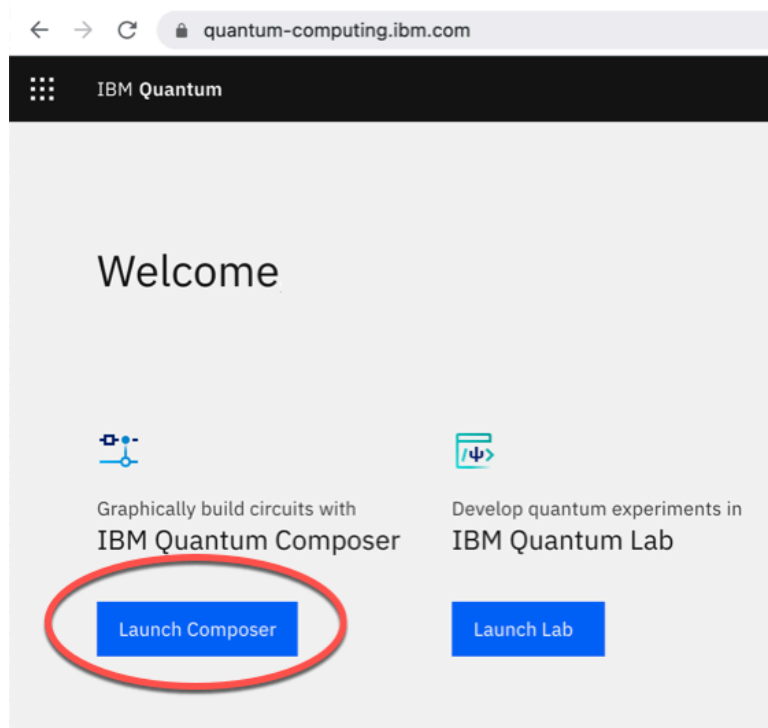
OPENQASM 2.0;
include "qelib1.inc";
gate statepreparation_0bktuj q0,q1,q2,q3 { h q0; h q1; h q2; ry(3.1816033)
q2; cry(3.8167242) q2,q1; x q2; cry(2.7468015) q2,q1; x q1; x q2; ccx
q1,q2,q3; cry(3.5363838) q3,q0; ccx q1,q2,q3; x q1; x q2; ccx q1,q2,q3;
cry(2.8601126) q3,q0; ccx q1,q2,q3; x q1; x q2; x q2; ccx q1,q2,q3;
cry(2.7571959) q3,q0; ccx q1,q2,q3; x q1; x q2; }
qreg q[4];
creg c[4];
statepreparation_0bktuj q[3],q[2],q[1],q[0];
barrier q[0],q[1],q[2],q[3];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
```

That's it! You've generated a fully working quantum circuit for state preparation

Test the circuit

To check that the circuit met our requirements, we'll use the IBM Quantum Experience site.

Login into the quantum computing site at <http://quantum-computing.ibm.com> (you can create a free account if you don't have one) and then launch the *IBM Quantum Composer*



Now copy the Qasm code generated by Classiq into the rightmost panel of the IBM Quantum Composer

The screenshot displays the Qiskit Composer interface. On the left, a file explorer shows 'Composer files' with a list of projects. The main workspace shows a quantum circuit titled 'Testing the Classiq output'. The circuit diagram includes 5 qubits (q0 to q4) and various gates: RY, CNOT, and measurements. The right panel shows the OpenQASM 2.0 code, which includes a state preparation routine and measurements. The bottom panels show the probability distribution and state vector.

OpenQASM 2.0

```

1 // Generated by Classiq.
2 // Creation timestamp: 2021-05-
3 // Random seed: 1875515605
4
5 OPENQASM 2.0;
6 include "qelib1.inc";
7 gate statepreparation_calg81
8   q0,q1,q2,q3 { h q0; h q1; h q2;
9   ry(3.1816033) q2;
10  cry(3.8167242) q2,q1; x q2;
11  cry(2.7468015) q2,q1; x q1; x
12  q2; ccx q1,q2,q3;
13  cry(3.5363838) q3,q0; ccx
14  q1,q2,q3; x q1; x q2; ccx
15  q1,q2,q3; cry(2.8601126) q3,q0;
16  ccx q1,q2,q3; x q1; x q2; x q2;
17  ccx q1,q2,q3; cry(2.7571959)
18  q3,q0; ccx q1,q2,q3; x q1; x
19  q2; }
20
21 qreg q[5];
22 creg c[5];
23 ry(1.885) q[0];
24 statepreparation_calg81
25   q[4],q[3],q[2],q[1];
26 barrier
27   q[0],q[1],q[2],q[3],q[4];
28 measure q[0] -> c[0];
29 measure q[1] -> c[1];
30 measure q[2] -> c[2];
31 measure q[3] -> c[3];
32 measure q[4] -> c[4];

```

Probabilities

Statevector

Computational basis states

Output state

Probability (%)

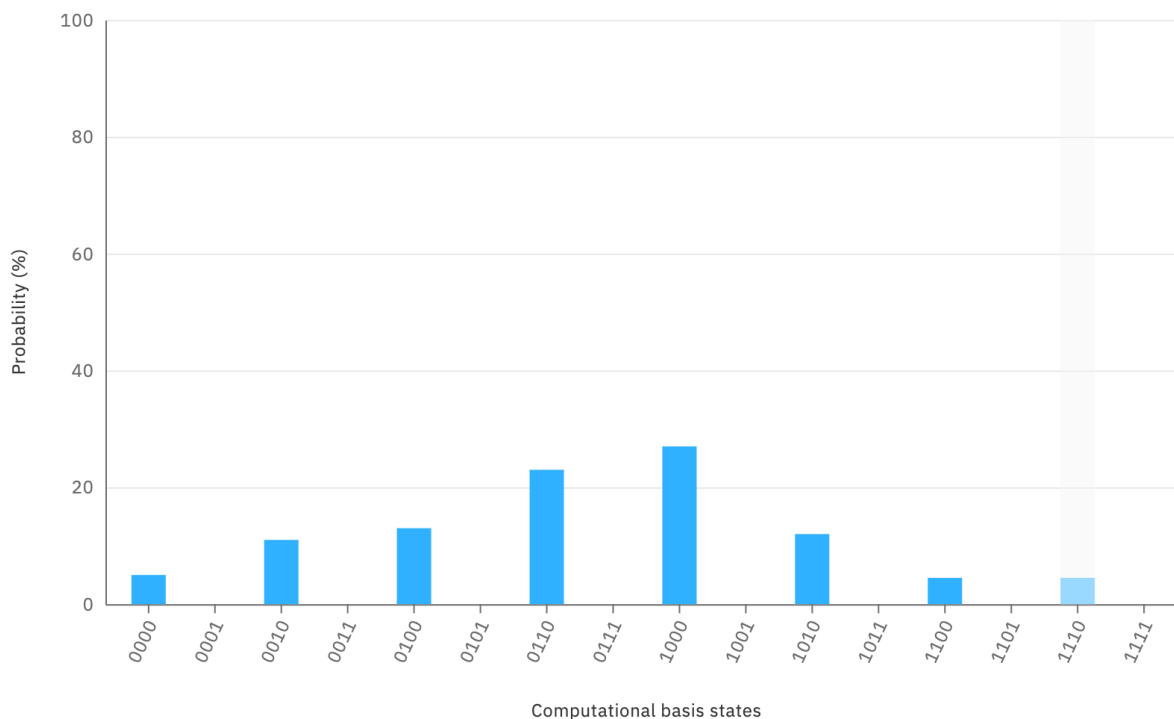
Amplitude

Computational basis states

Output state

Show more

The 'measure' lines at the end of the code force a measurement, but if we want to see the probability distribution, we can remove them, where we will get something like:



Note: To see a ‘bell-shaped’ distribution sorted in the proper order, you might need to add a swap command such as

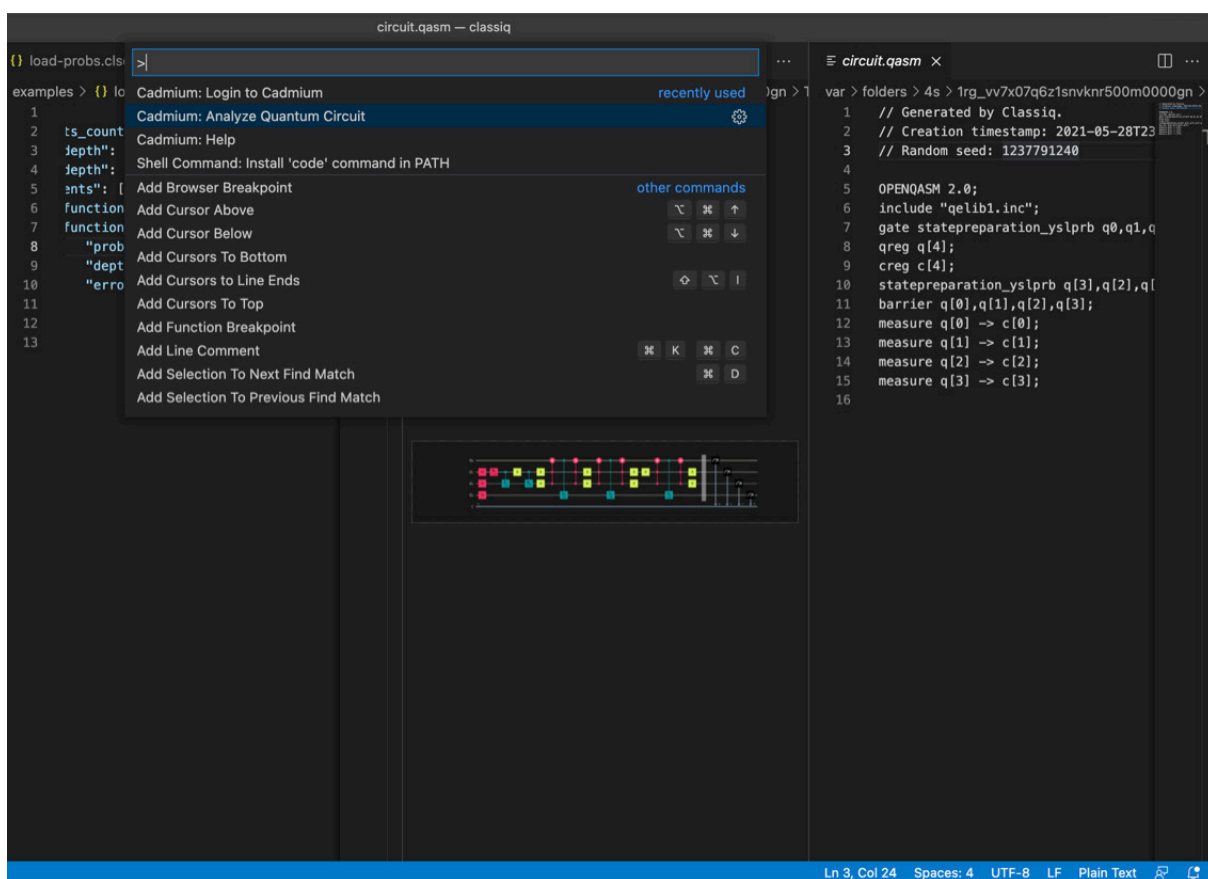
```
swap q[1],q[3];
```

Congratulations!

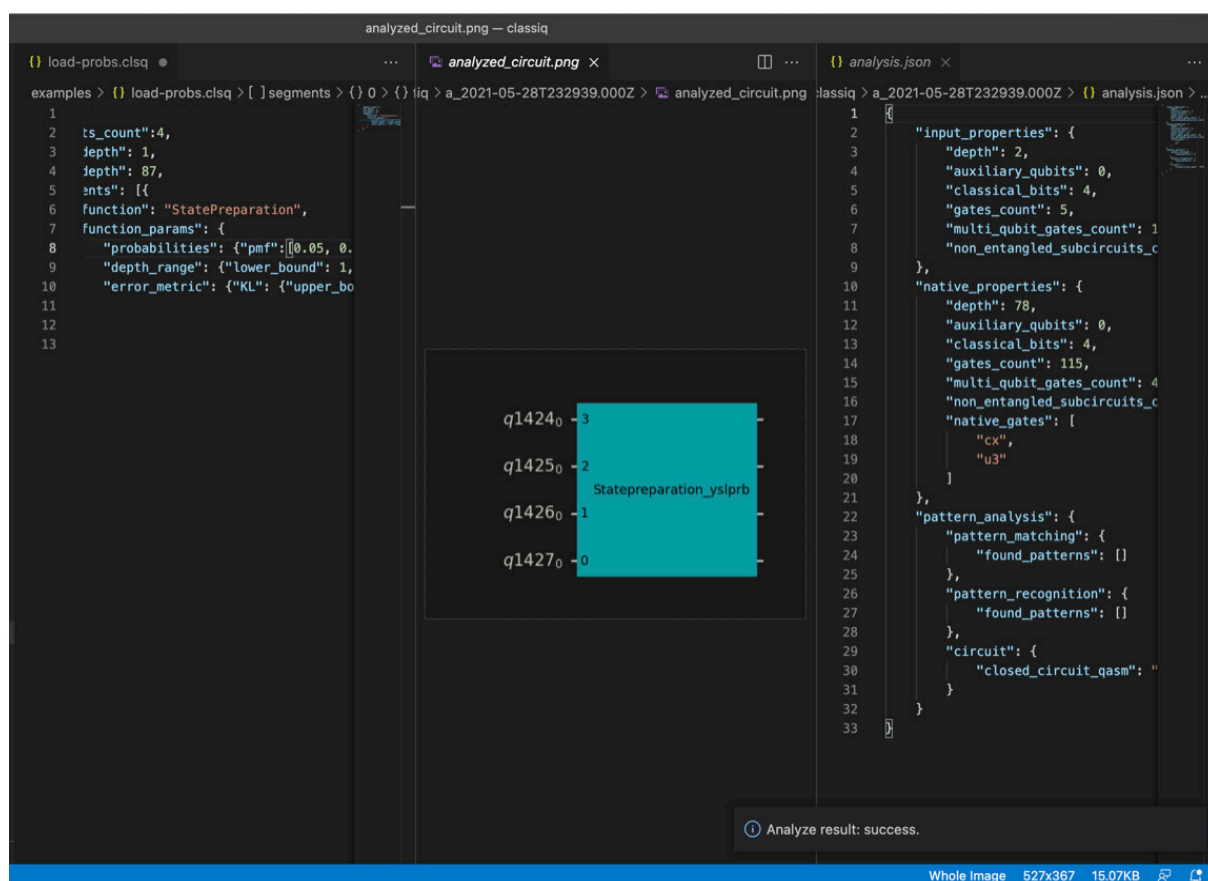
You’ve just completed a working quantum circuit with the Classiq platform. What else can we learn about this circuit and how can we change it?

Analyzing the circuit

You can learn important attributes about the circuit by analyzing it. Once a circuit has is ready, click on the Qasm code (rightmost panel) and then select “Analyze Quantum Circuit” from the command palette.



The result of the analysis will look something like this:



Let's focus on the analysis result on the right side. You can see that the analyzer provided many interesting details, such as the depth of the circuit, the number of gates being used, and the types of gates that were chosen for this circuit. These are highlighted in the capture below

```

1 {
2   "input_properties": {
3     "depth": 2,
4     "auxiliary_qubits": 0,
5     "classical_bits": 4,
6     "gates_count": 5,
7     "multi_qubit_gates_count": 1,
8     "non_entangled_subcircuits_count": 1
9   },
10  "native_properties": {
11    "depth": 78,
12    "auxiliary_qubits": 0,
13    "classical_bits": 4,
14    "gates_count": 115,
15    "multi_qubit_gates_count": 46,
16    "non_entangled_subcircuits_count": 1,
17    "native_gates": [
18      "cx",

```

```

19     "u3"
20   ],
21 },
22 "pattern_analysis": {
23   "pattern_matching": {
24     "found_patterns": []
25   },
26   "pattern_recognition": {
27     "found_patterns": []
28   },
29   "circuit": {
30     "closed_circuit_qasm": "OPENQASM 2.0;\ninclude \"qelib1.inc\";\ngate st
31   }
32 }
33 }

```

Modifying the code

In our model, you might remember that we defined the desired states and defined the desired accuracy that we expect of the state preparation. In this case - highlighted below - we asked for 1% accuracy.

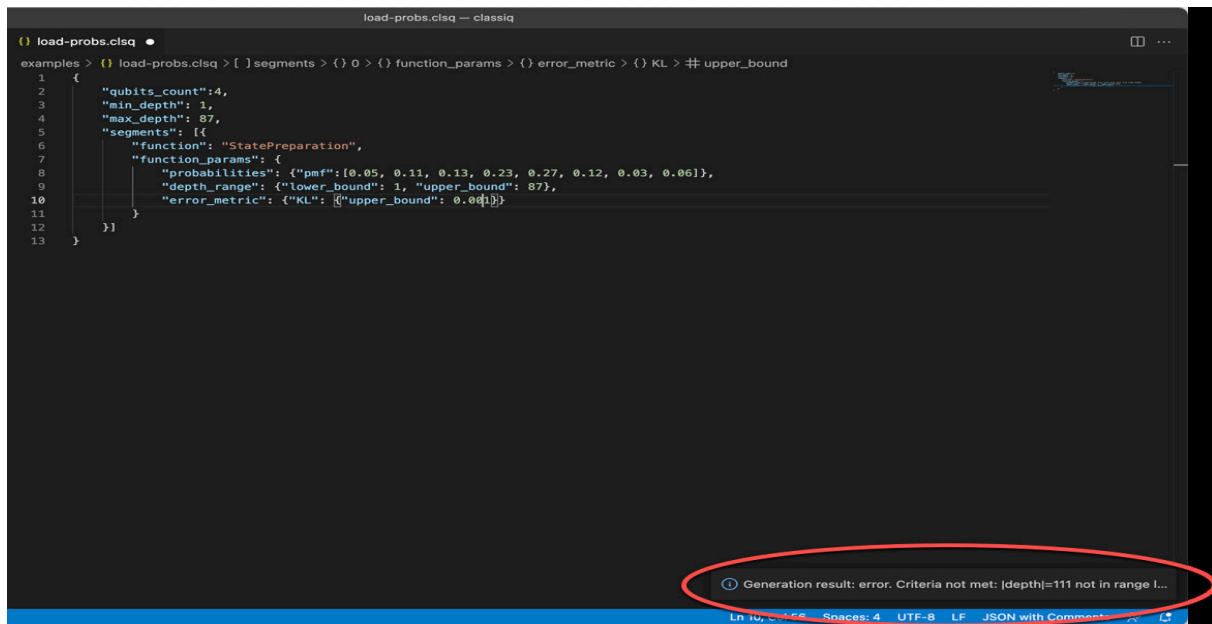
```

1 {
2   "qubits_count": 4,
3   "min_depth": 1,
4   "max_depth": 87,
5   "segments": [{
6     "function": "StatePreparation",
7     "function_params": {
8       "probabilities": {"pmf": [0.05, 0.11, 0.13, 0.23, 0.27, 0.12, 0.03, 0.06]},
9       "depth_range": {"lower_bound": 1, "upper_bound": 87},
10      "error_metric": {"KL": {"upper_bound": 0.01}}
11    }
12  ]
13 }

```

But what if we need greater accuracy? For this, we can try changing “*upper bound*” parameter of “*error_metric*” from its current value of 0.01 to something else, such as 0.001.

Try it! Change the accuracy and then regenerate the circuit. Interestingly, you'll get an error message (in the bottom right corner of the screen)



The screenshot shows the Classiq IDE interface. The main editor displays a JSON configuration for a quantum circuit. The configuration includes parameters like qubits_count, min_depth, max_depth, and a function named StatePreparation. The error message at the bottom right, circled in red, states: "Generation result: error. Criteria not met: |depth|=111 not in range I...".

```

load-probs.clsq — classiq
examples > {} load-probs.clsq > [ ] segments > {} 0 > {} function_params > {} error_metric > {} KL > # upper_bound
1
2 {
3   "qubits_count": 4,
4   "min_depth": 1,
5   "max_depth": 87,
6   "segments": [{}
7     "function": "StatePreparation",
8     "function_params": {
9       "probabilities": {"pmf": [0.05, 0.11, 0.13, 0.23, 0.27, 0.12, 0.03, 0.06]},
10      "depth_range": {"lower_bound": 1, "upper_bound": 87},
11      "error_metric": {"KL": {"upper_bound": 0.01}}
12    ]
13  }

```

Generation result: error. Criteria not met: |depth|=111 not in range I...

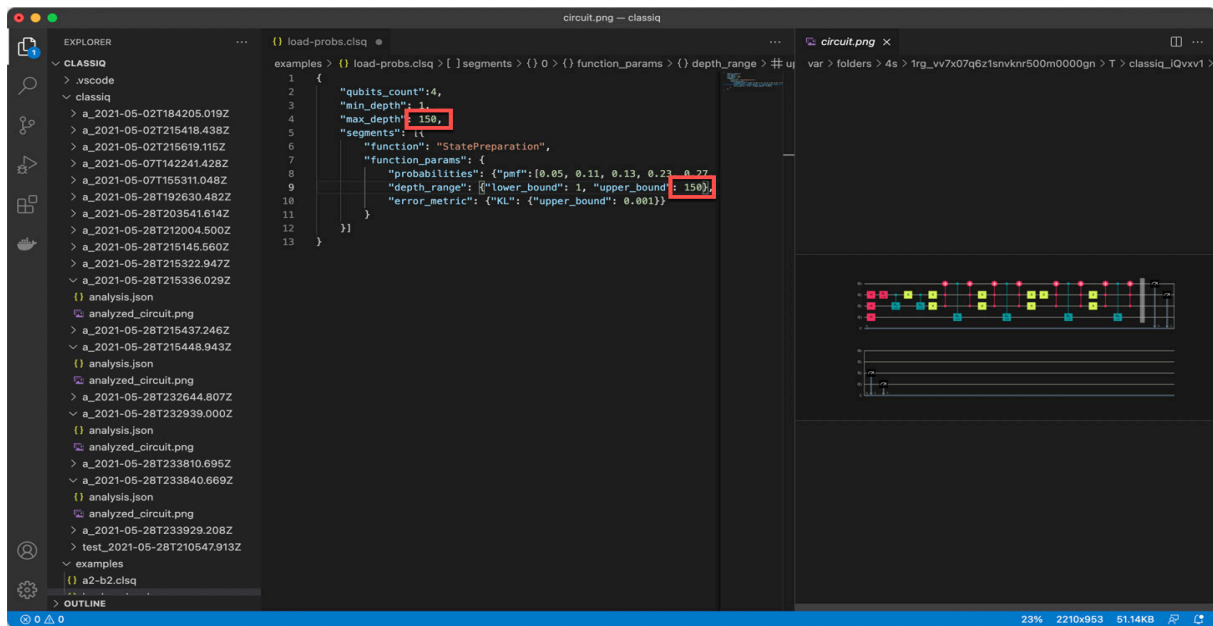
Ln 10, Col 10 Spaces: 4 UTF-8 LF JSON with Comments

Overcoming a constraint

What just happened?

The circuit generator looks at all the constraints given in the model, and one of the constraints was the maximum depth of the circuit. When the accuracy is 0.01, the analyzer revealed to us that the circuit depth was 78. But to reach greater accuracy, we need greater depth and the current model set 87 as the *upper_bound* of the depth, which might not be enough. The upper limit might be a limitation of the target hardware, but if this limitation is arbitrary, let's increase the depth to 150 and see what happens.

This time, the circuit generation was successful:



Note: I closed the Qasm panel just so that we can see the circuit better.

Analyze the circuit once again, and now we see that the resulting circuit has a depth of 100, which is within the bounds that we set



Try another optimization

Perhaps even with the lower accuracy, this circuit has too many gates. To explore the alternatives, change the accuracy to 0.1 and see what happens.

Congratulations!

In this tutorial, you successfully analyzed a circuit, changed it, and learned to overcome some constraints. We hope you had a chance to appreciate how easy it is to set constraints, explore alternatives, and quickly get to a working quantum circuit with the Classiq platform.



REVOLUTIONIZING THE DEVELOPMENT OF QUANTUM SOFTWARE

In this tutorial, you successfully analyzed a circuit, changed it, and learned to overcome some constraints. We hope you had a chance to appreciate how easy it is to set constraints, explore alternatives, and quickly get to a working quantum circuit with the Classiq platform.

REQUEST A
DEMO TODAY

hello@classiq.io
www.classiq.io

