Introduction to QSim

qctoolkit.in

Vivek N (C-DAC)

What it Offers?

- **QSim** is a robust QC Simulator integrated with a GUI based Workbench

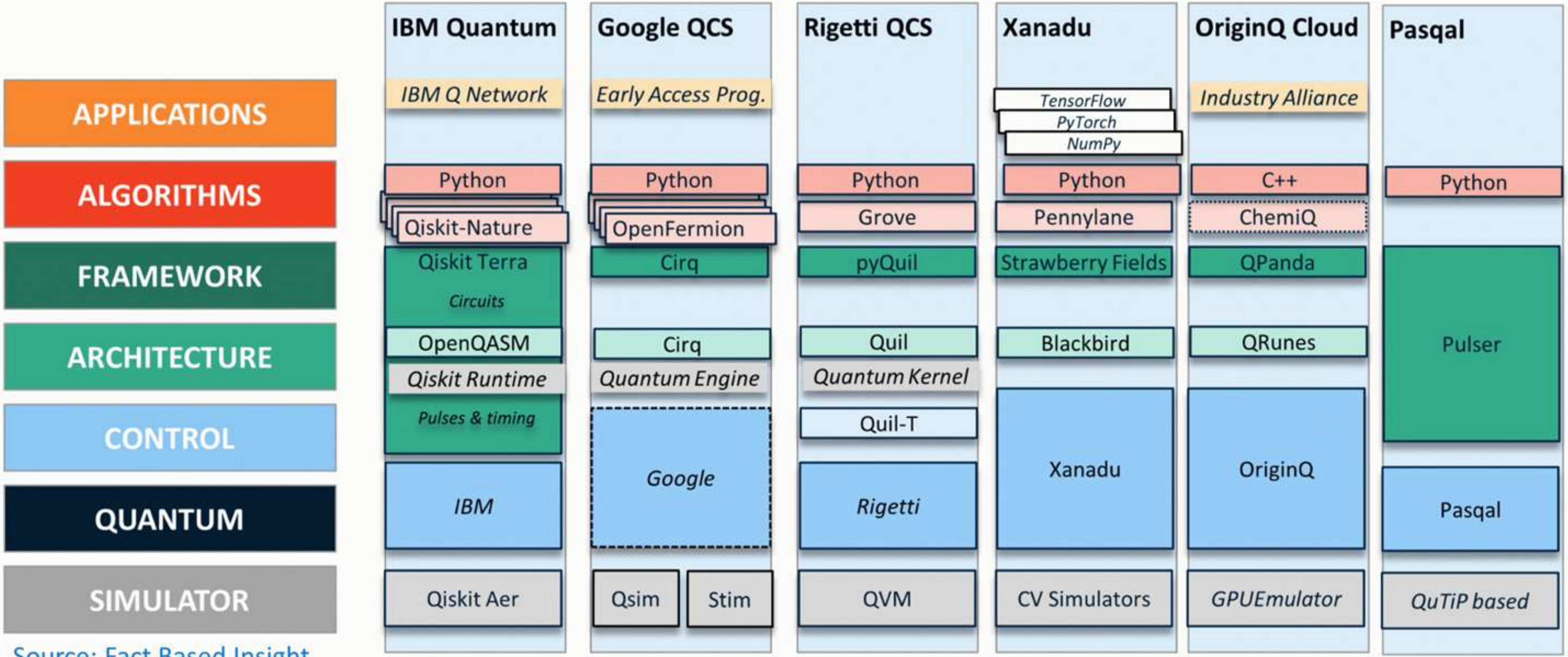
- Students / Researchers can create Quantum Circuits and Quantum Programs and view the simulated outputs

- QSim simulator is an open-source software written in Python, which is added as a new backend to IBM's Qiskit platform
- It extends the existing Qiskit capability, while retaining the convenience (e.g. portability, documentation, graphical interface) of the Qiskit format
- Simulator : arxiv.org/abs/1908.05154
- https://github.com/indian-institute-of-science-qc/qiskit-aakash

  Latest updates on terra_upgrade branch of repo
- https://mybinder.org/v2/gh/indian-institute-of-science-qc/qiskit-aakash/terra_upgrade

https://arxiv.org/abs/1908.05154

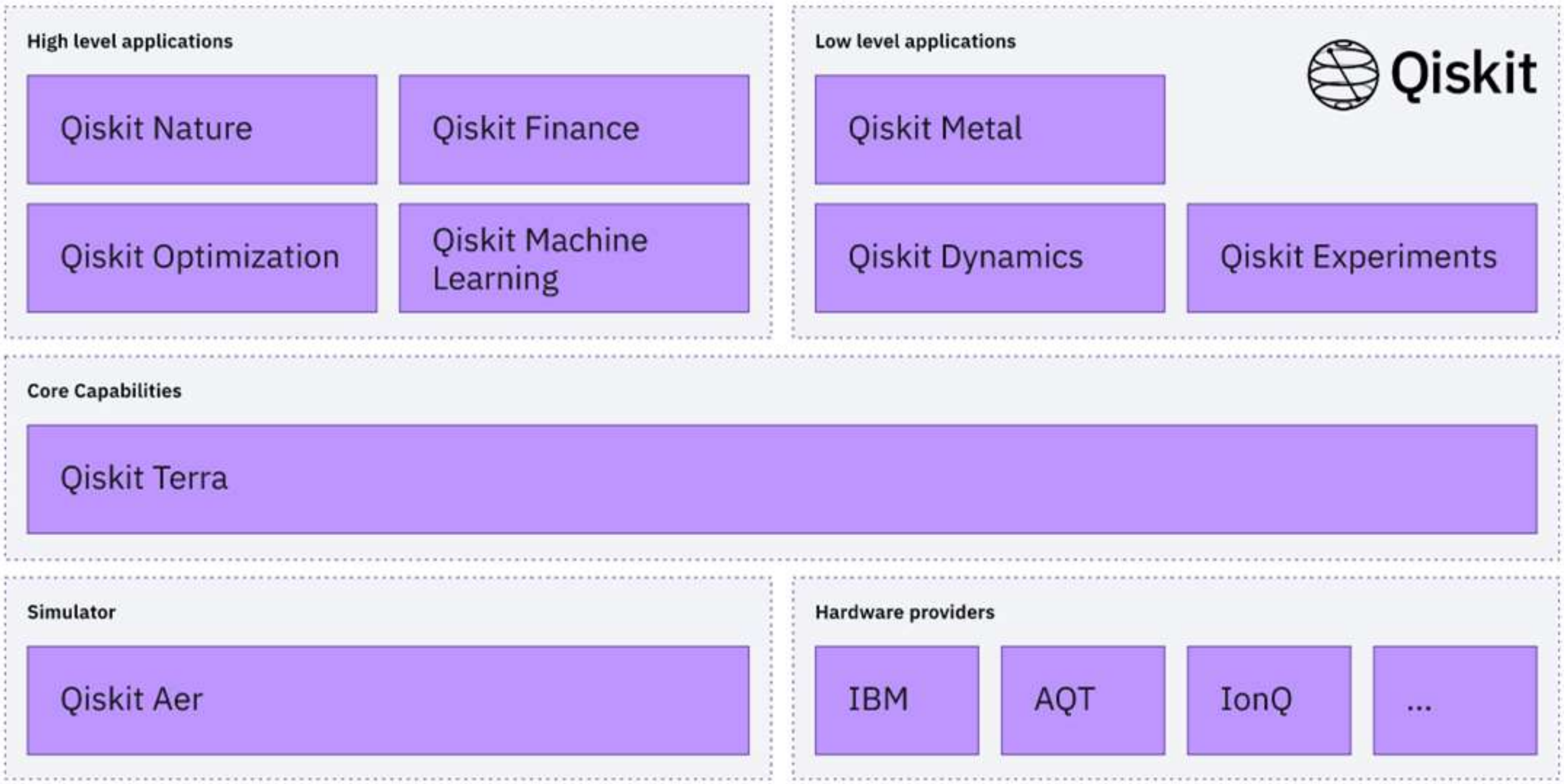
Early gate-model full-stack players



Source: Fact Based Insight

https://quantumcomputingreport.com/quantum-software-outlook-2022/

Qiskit is open-source software for working with quantum computers at the level of circuits, pulses, and algorithms



https://qiskit.org/documentation/

pip install qiskit

Qiskit Software	Version
`qiskit-terra`	0.22.2
`qiskit-aer`	0.11.1
`qiskit-ibmq-provider`	0.19.2
`qiskit`	0.39.2
`qiskit-nature`	0.4.5
`qiskit-finance`	0.3.4
`qiskit-optimization`	0.4.0
`qiskit-machine-learning`	0.4.0
System information	
Python version	3.8.14
Python compiler	GCC 9.4.0
Python build	default, Sep 7 2022 14:28:32
OS	Linux
CPUs	2
Memory (Gb)	6.78125
	Fri Nov 04 02:27:15 2022 UTC

When using Qiskit a user workflow nominally consists of following four high-level steps:

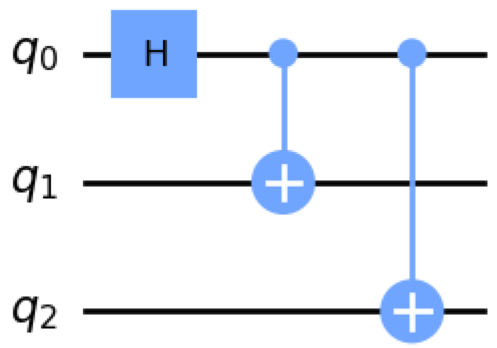
- **Build**: Design a quantum circuit(s) that represents the problem you are considering.
- **Compile**: Compile circuits for a specific quantum service, e.g. a quantum system or classical simulator.
- **Run**: Run the compiled circuits on the specified quantum service(s). These services can be cloud-based or local.
- **Analyze**: Compute summary statistics and visualize the results of the experiments.

The basic element needed for your first program is the **QuantumCircuit**
This is taken care by the **Qiskit Terra**

```
import numpy as np
from qiskit import *
```

```
# Create a Quantum Circuit acting on a quantum register of three qubits
circ = QuantumCircuit(3)
```

```
# Add a H gate on qubit 0, putting this qubit in superposition.
circ.h(0)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 1, putting
# the qubits in a Bell state.
circ.cx(0, 1)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 2, putting
# the qubits in a GHZ state.
circ.cx(0, 2)
```



Simulating circuits using **Qiskit Aer**

```
# Import Aer
from qiskit import Aer

# Run the quantum circuit on a statevector simulator backend
backend = Aer.get_backend('statevector_simulator')
```

```
# Create a Quantum Program for execution
job = backend.run(circ)
```

```
outputstate = result.get_statevector(circ, decimals=3)
print(outputstate)
```

```
Statevector([0.707+0.j, 0.   +0.j, 0.   +0.j, 0.   +0.j, 0.   +0.j,
             0.   +0.j, 0.   +0.j, 0.707+0.j],
            dims=(2, 2, 2))
```

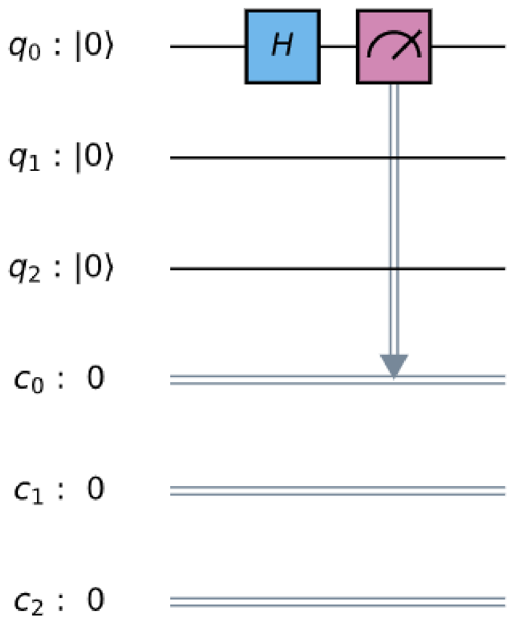
1 - Program on QSim /Density Matrix simulator

```
from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, execute, BasicAer
import numpy as np
```

```
#Options & Noise goes here - Don't change options variable name & block
options = {

"rotation_error": {'rx':[1.0, 0.0], 'ry':[1.0, 0.0], 'rz':[1.0,0.0]},
"tsp_model_error": [1.0, 0.0],
"thermal_factor": 1.0,
"decoherence_factor": 1.0,
"depolarization_factor": 1.0,
"bell_depolarization_factor": 1.0,
"decay_factor": 1.0,
}
```

2 - Circuit Creation same as provided by Qiskit Terra

```
qc = QuantumCircuit()
q = QuantumRegister(3, 'q')
c = ClassicalRegister(3, 'c')

qc.add_register(q)
qc.add_register(c)

qc.h(q[0])
qc.measure(q[0], c[0])
```



3 – Running Quantum Circuit on QSim and Getting Results

```
backend = BasicAer.get_backend('dm_simulator')
job = execute(qc, backend=backend, **options)
job_result = job.result()
print(job_result.results[0].data.densitymatrix)
```

Implementation

- Standard formulation of quantum mechanics, states are vectors in a Hilbert space and evolve by unitary transformations, $|\psi> \to U|\psi>$. This evolution is deterministic, continuous and reversible.
- It is appropriate for describing the pure states of a closed quantum system, but is insufficient for describing the mixed states that result from interactions of an open quantum system with its environment
- The most general description of a quantum system is in terms of its density matrix ρ

https://arxiv.org/abs/1908.05154

- Quantum systems are highly sensitive to disturbances from the environment; even necessary controls and observations perturb them.
- The available, and upcoming, quantum devices are noisy, and techniques to bring down the environmental error rate are being intensively pursued
- It is necessary to come up with error-resilient system designs, as well as techniques that validate and verify the results

https://arxiv.org/abs/1908.05154

- This era of noisy intermediate scale quantum systems has been labeled **NISQ**
- Such systems are often special purpose platforms, with limited capabilities
- They roughly span devices with 10- 100 qubits, 10-1000 logic operations, limited interactions between qubits, and with no error correction since the fault-tolerance threshold is orders of magnitudes away

- A quantum computation may suffer from many sources of error
    - Imprecise initial state preparation
    - Imperfect logic gate implementation
    - Disturbances to the data in memory
    - Error-prone measurements

So a realistic quantum simulator would have to include all of them with appropriate probability distributions

https://arxiv.org/abs/1908.05154

QSim: Quantum Computer Simulator Toolkit

QSim
QC Workbench

Simulate quantum circuits: Simulation of Quantum circuits with custom parameters.

Quantum Noise: Realistic simulator considering effects of noise

Intuitive UI: Intuitive UI/UX helps users to conceptualize and create quantum programs

Examples & Help: Online help, solved examples and learning material.

Secured user management: Secure user management with options to save quantum programs/circuits

Code editor: Advanced Python code editor for Quantum Circuits.

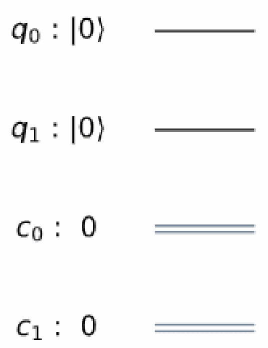
QC Workbench Features

- **Interactive Python code editor with IDE features such as**
  - Syntax highlighting
  - Parsing and error checking
  - Autocompleting keywords and variables
  - Code folding and unfolding
  - Automatic braces detecting and closing
  - Search and replace keywords.
- **Build and visualize Quantum Circuits**

</> Python Editor

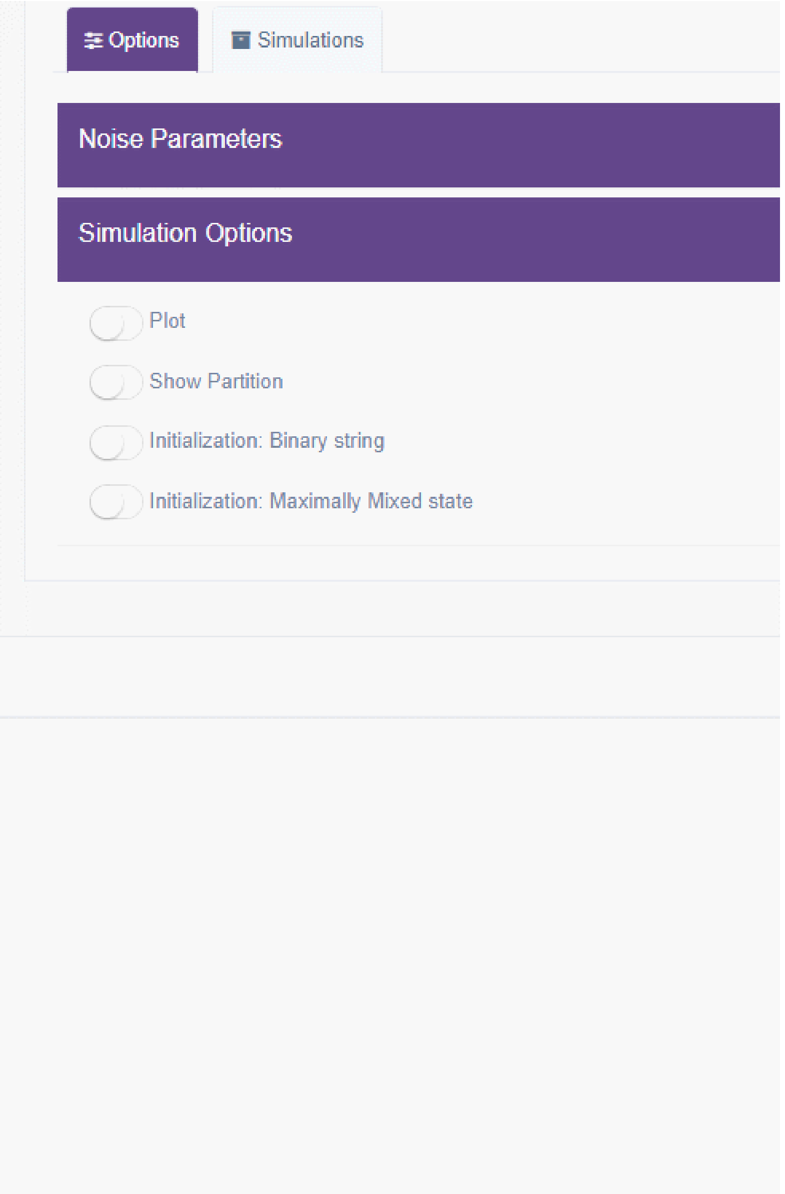
```
"tsp_model_error": [1.0, 1.0],
"thermal_factor": 1.0,
"decoherence_factor": 1.0,
"depolarization_factor": 1.0,
"bell_depolarization_factor": 1.0,
"decay_factor": 1.0,
}
#####Write your code after this line######
qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q,creg_c)

backend = BasicAer.get_backend('dm_simulator')
job = execute(circuit, backend=backend, **options)
job_result = job.result()
```

</> Quantum Circuit



QC Workbench Features

- **Induce Noise parameters**
  - Rotation Error
  - TSP Error
  - Decay factor
  - Decoherence
  - Depolarization Factor
  - Thermal Factor
  - Bell Depolarization Factor
- **Customize simulation options**
  - Enable/Disable Plot
  - Enable/Disable circuit partioning
  - Circuit initialization options.



QC Workbench Features

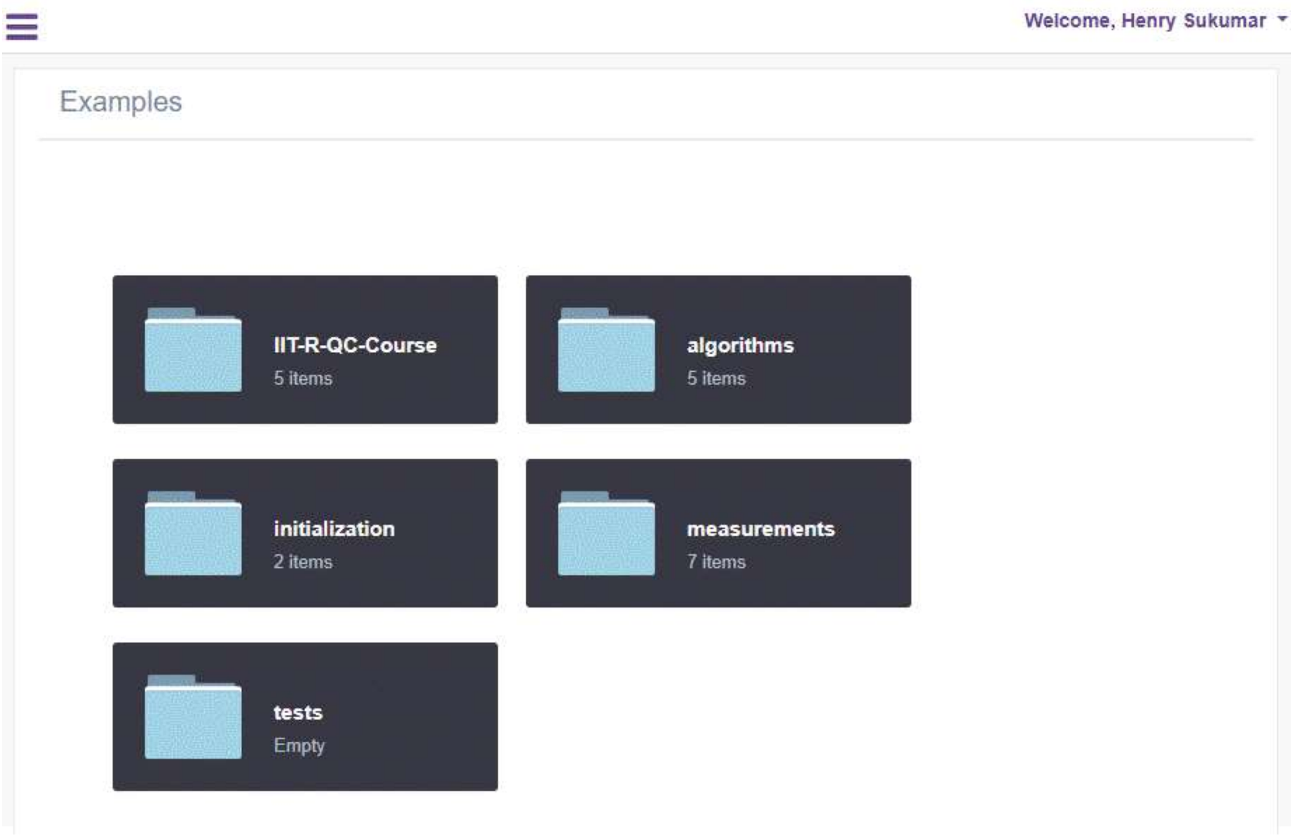
- **Support for multiple measurement options**
  - Single QuBit measurement
  - Expectation measurement
  - Ensemble measurement

</> Python Editor

```
#####write your code after this line######
qc = QuantumCircuit()
q = QuantumRegister(1, 'q')
c1 = ClassicalRegister(2, 'c1')

qc.add_register(q)
qc.add_register(c1)

qc.h(q[0])
qc.measure(q[0], c1[0])

backend = BasicAer.get_backend('dm_simulator')
job = execute(qc, backend=backend, **options)
job_result = job.result()
print(job_result['results'][0]['data']['densitymatrix'])
```

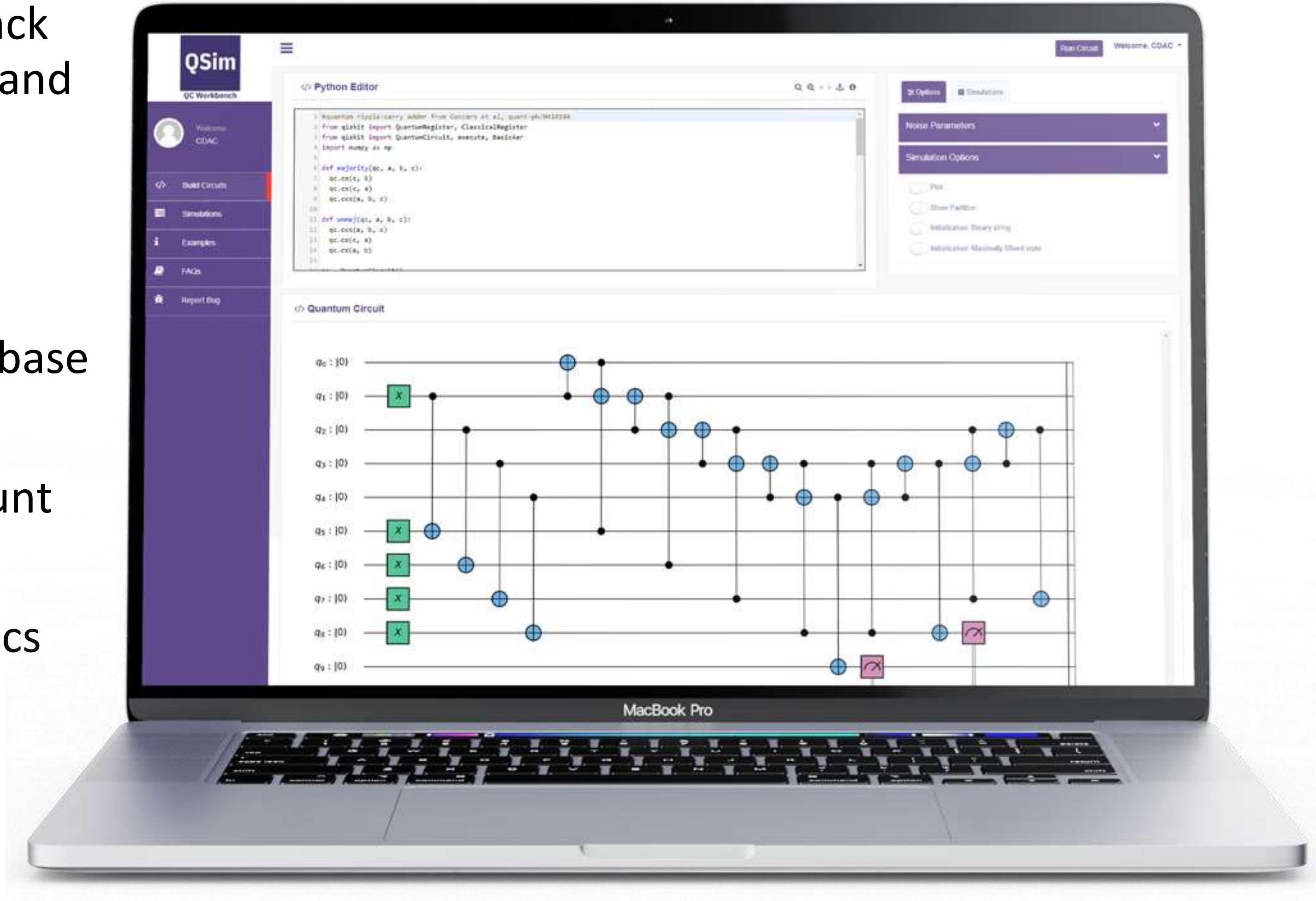
QC Workbench Features

- **Pre-loaded Quantum examples and algorithms**
    - Deutsch-Jozsa Algorithm
    - Hubbard model
    - QFT
    - Grover's algorithm
    - Ripple Carry Adder

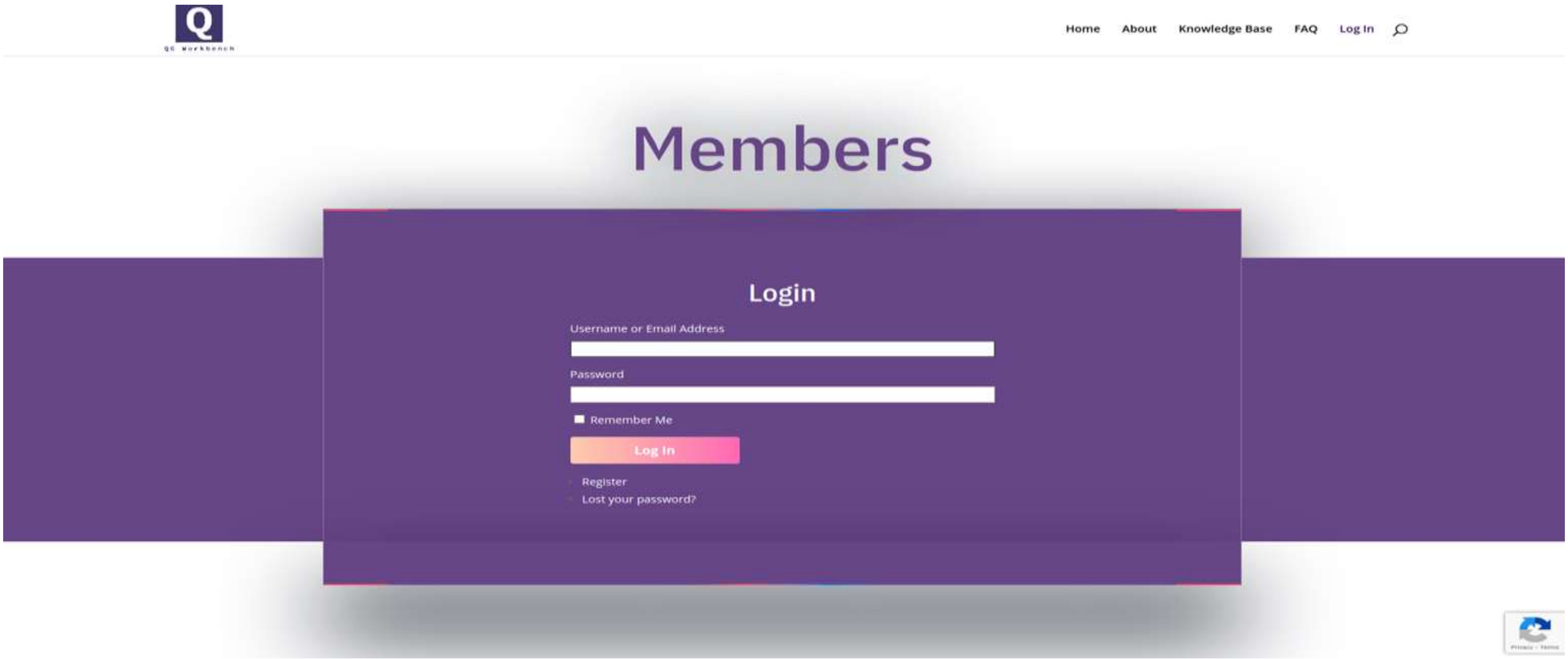


QC Workbench Features

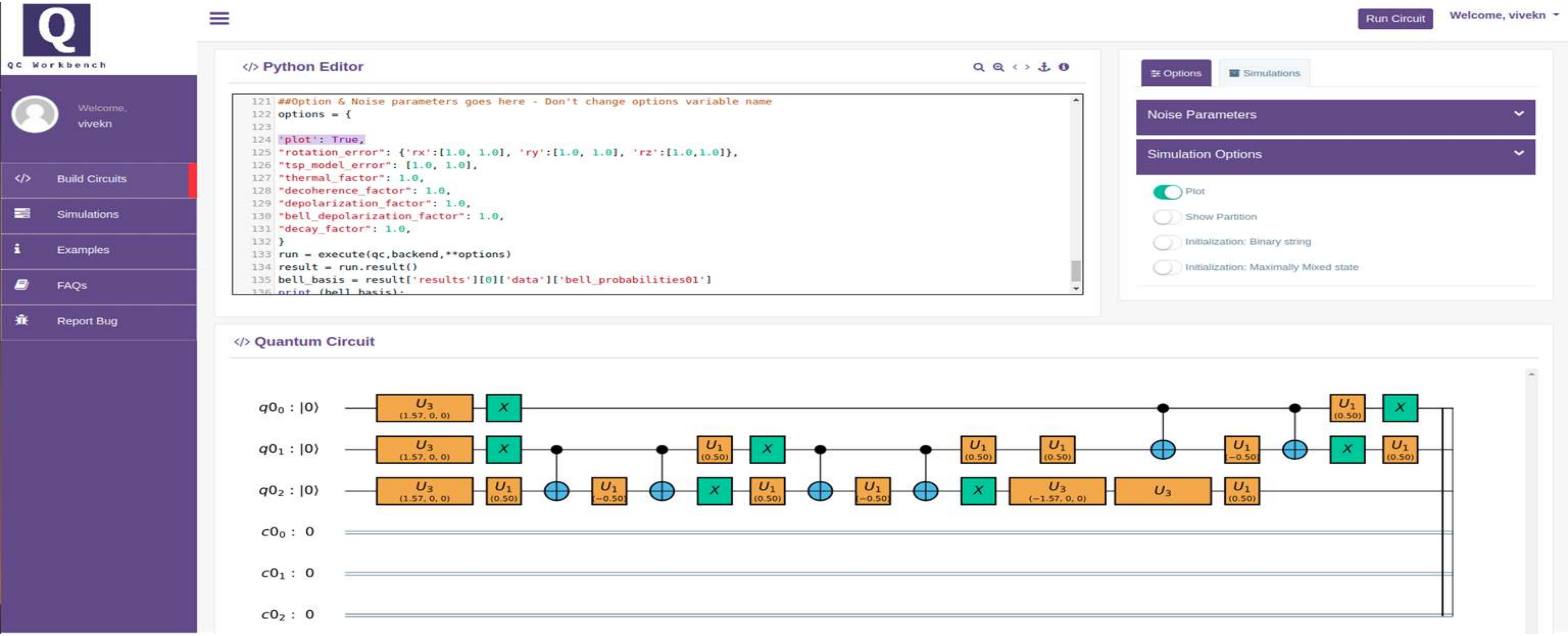
- Submit simulations, track progress, fetch results and accounting
- Plot and visualization histograms
- Integrated Knowledge base & Guides
- Secure Login and account management
- User simulation statistics
- Bug reporting.



Login Screen (qctoolkit.in)



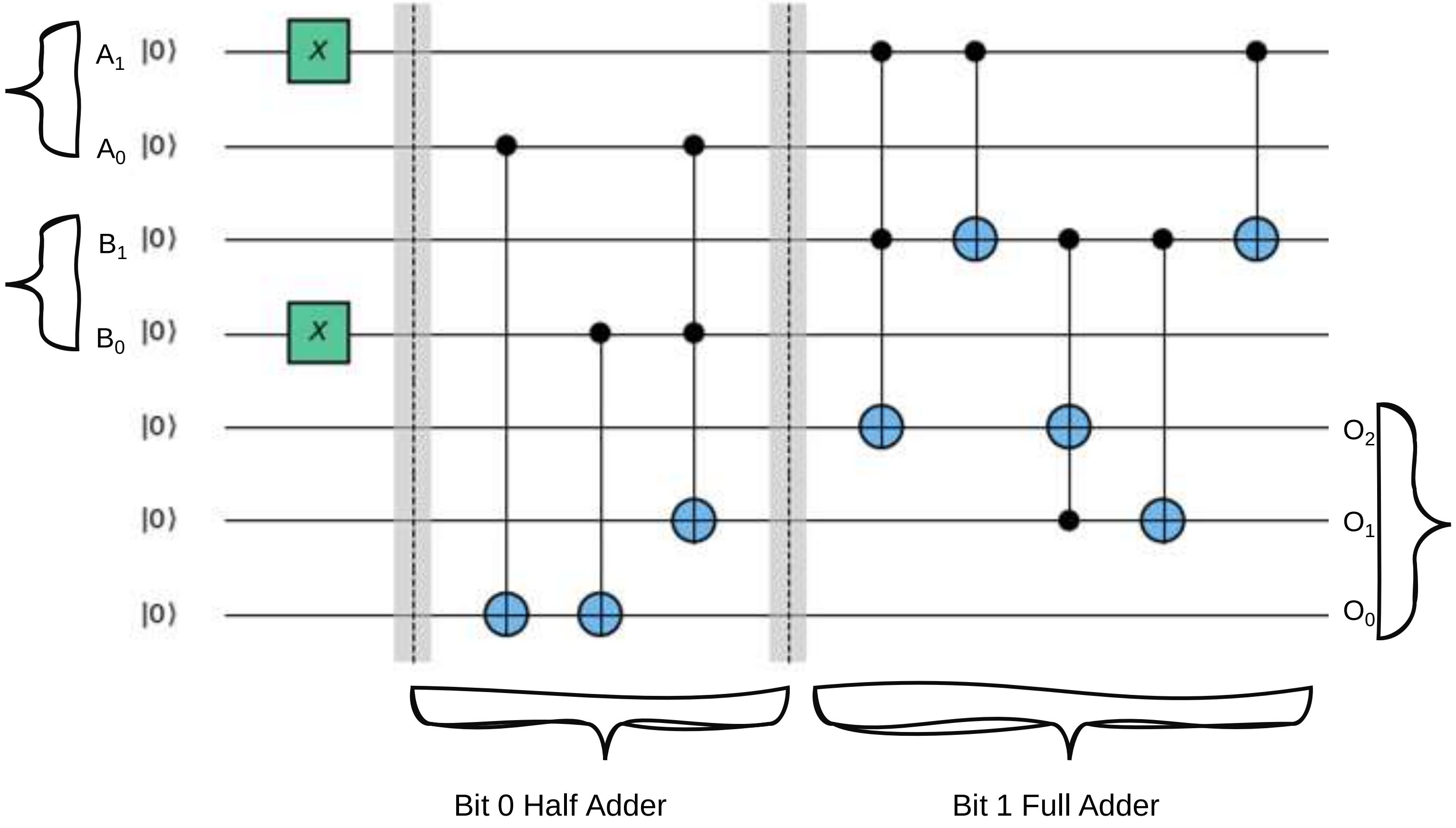
Interactive UI



Quantum Circuit for 2-bit Ripple Adder

Input A: 2 = $(10)_2$

Input B: 1 = $(01)_2$



Output: 3 = $(11)_2$

Credits IITR

Country	Technology	Companies/ University
The United States of America	**Superconducting based**	IBM, Google, Rigetti
	Ion Trap-based	IonQ
	Photonics based	Psi- Quantum
	Neutral atoms based	ColdQuanta, Atom Computing QuEra Computing
	Semiconductor based	Princeton University
Canada	**Photonics based**	Xanadu
The United Kingdom	**Ion Trap- based**	Quantinuum, Universal Quantum
	Photonics based	Orca Computing, Tundrasystems Global
	Semiconductor based	Quantum Motion
Finland	**Superconducting based**	IQM
	Semiconducting based	QuTech
Germany	**Ion Trap- based**	eleQtron
	Neutral atom based	Planqc
Austria	**Ion trap- based**	Alpine Quantum Technologies
France	**Neutral atoms based**	PasQal
	Photonics based	Quandela

Links

- Essence of Linear Algebra

  https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab

- Binder Image for the latest QSim(density matrix simulator)

  https://mybinder.org/v2/gh/indian-institute-of-science-qc/qiskit-aakash/terra_upgrade?labpath=dm_simulator_user_guide%2Fuser_guide.ipynb

- Density Matrix

  https://learn.qiskit.org/v1/course/quantum-hardware/density-matrix