Classiq Execution

Execution with Classiq

- Use ExecutionSession() to work within a python editor.
  - Also creates a job on the Classiq server.
- `es = ExecutionSession(qprog, [ExecutionPreferences])`
  - Arguments inside “[]” are optional
  - ExecutionPreferences(num_shots, ...)
- Run ExecutionSession using methods such as
  - `es.sample()`
    - to obtain an ExecutionDetails object
  - `es.estimate(Hamiltonian)`
  - The Hamiltonian is a list of PauliTerm
    - `[PauliTerm(pauli, coefficient), ...,]`

```
"""
ExecutionDetails(
   vendor_format_result={},
   counts={"0": 1045, "1": 1003},
   counts_lsb_right=True,
   probabilities={},
   parsed_states={"0": {"q": [0]}, "1": {"q":
[1]}},
   histogram=None,
   output_qubits_map={"q": (0,)},
   state_vector=None,
   parsed_state_vector_states=None,
   physical_qubits_map={"q": (0,)},
   num_shots=2048,
)
"""
```

Execution with Parameters

- Some Quantum Programs require parameters as input.
  - e.g., ansatz circuits requires a list of angles of rotation.
- Run ExecutionSession with ExecutionParams.
  - `.sample(ExecutionParams)`
  - `.estimate(Hamiltonian, ExecutionParams)`
- ExecutionParams objects are special dictionaries.
  - where the Keys are 'params_n'.

execution_params =
{'params_0': 3.141592653589793,
 'params_1': 1.5707963267948966,
 'params_2': 1.0471975511965976,
 'params_3': 0.7853981633974483,
 'params_4': 0.6283185307179586,
 'params_5': 0.5235987755982988,
 'params_6': 0.4487989505128276,
 'params_7': 0.3926990816987241 4,
 'params_8': 0.3490658503988659}

Integration with Qiskit (and others)

- Exporting quantum circuit from Classiq
- in QASM2 format (and other formats)
- importing into Qiskit (and others)

- Avoid sending data to Classiq server
- Simulate locally using Qiskit Aer Simulator library

```
# Exporting
qmod = create_model(main)
qprog = synthesize(qmod)
qcode = qprog.to_program()

# Importing from QuantumCode
circ = qiskit.qasm2.loads(qcode.code)

# Importing from *.qasm files
circ = qiskit.qasm2.load(“file_path”)

# Visualization
display(circ.draw("mpl"))

# To breakdown custom gates
display(circ.decompose().draw(“mpl”))
```