In [3]:

```
# Bernstein-Vazirani algorithm in Cirq
```

In [128]:

```
# Print version of SDK
!pip show amazon-braket-sdk | grep Version

# Import Braket libraries
from braket.circuits import Circuit, Gate, Moments
from braket.circuits.instruction import Instruction
from braket.aws import AwsDevice
import matplotlib.pyplot as plt
import boto3
import time
import cirq
import random
import numpy as np
```

```
Version: 0.6.0
```

In [129]:

```
# Function to run quantum task, check the status thereof and collect results
def get_result(device, circ, s3_folder):
```

```python
    # get number of qubits
    num_qubits = circ.qubit_count

    # specify desired results_types
    circ.probability()

    # submit task: define task (asynchronous)
    if device.name == 'DefaultSimulator':
        task = device.run(circ, shots=1000)
    else:
        task = device.run(circ, s3_folder,
                          shots=1000,
                          poll_timeout_seconds=1000)

    # Get ID of submitted task
    task_id = task.id
#     print('Task ID :', task_id)

    # Wait for job to complete
    status_list = []
    status = task.state()
    status_list += [status]
    print('Status:', status)

    # Only notify the user when there's a status change
    while status != 'COMPLETED':
        status = task.state()
        if status != status_list[-1]:
            print('Status:', status)
        status_list += [status]

    # get result
    result = task.result()

    # get metadata
    metadata = result.task_metadata

    # get output probabilities
    probs_values = result.values[0]

    # get measurment results
    measurement_counts = result.measurement_counts

    # print measurment results
    print('measurement_counts:', measurement_counts)

    # bitstrings
    format_bitstring = '{0:0' + str(num_qubits) + 'b}'
    bitstring_keys = [format_bitstring.format(ii) for ii in range(2**num_qubits)]

    # plot probabalities
    plt.bar(bitstring_keys, probs_values);
    plt.xlabel('bitstrings');
```

```
    plt.ylabel('probability');
    plt.xticks(rotation=90);
    plt.show()

    return measurement_counts
```

In [130]:

```
qubit_count = 8

# Number of times to sample from the circuit
circuit_sample_count = 3

# Choose qubits to use
input_qubits = [cirq.GridQubit(i, 0) for i in range(qubit_count)]
output_qubit = cirq.GridQubit(qubit_count, 0)

# Pick coefficients for the oracle and create a circuit to query it
secret_bias_bit = random.randint(0 ,1)
secret_factor_bits = [random.randint(0, 1) for _ in range(qubit_count)]
```

In [131]:

```
def make_oracle(input_qubits, output_qubit, secret_factor_bits, secret_bias_bit):
    # Gates implementing the function f(a) = a*factors + bias (mod1)
    if secret_bias_bit:
        yield cirq.X(output_qubit)

    for qubit, bit in zip(input_qubits, secret_factor_bits):
        if bit:
            yield cirq.CNOT(qubit, output_qubit)
```

In [132]:

```
oracle = make_oracle(input_qubits, output_qubit,
```

```
        secret_factor_bits, secret_bias_bit)
print('Secret function:\nf(x) = x*<{}> + {} (mod 2'.format(
    ','.join(str(e) for e in secret_factor_bits), secret_bias_bit))
```

```
Secret function:
f(x) = x*<0,1,0,1,1,0,0,0> + 0 (mod 2
```

In [133]:

```
"""make the bernstein vazirani circuit"""
def make_bernstein_vazirani_circuit(input_qubits, output_qubit, oracle):
    # Solves for factors in f(a) = a*factors + bias (mod2) with one query
    c = cirq.Circuit()

    # Initialize qubits
    c.append([
        cirq.X(output_qubit),
        cirq.H(output_qubit),
        cirq.H.on_each(*input_qubits)
    ])

    # Query oracle
    c.append(oracle)

    # Measure in X basis
    c.append([
        cirq.H.on_each(*input_qubits),
        cirq.measure(*input_qubits, key='result')
    ])

    return c
```

In [134]:

```
# Embed the oracle into a special quantum circuit querying it exactly once
cirq_circuit = make_bernstein_vazirani_circuit(input_qubits, output_qubit, oracle)
```

```python
print('\nCircuit:')
print(cirq_circuit)
```

```
Circuit:
(0, 0): ───H───H───────────────────M('result')───
                                   │
(1, 0): ───H───────@───H───────────M─────────────
                   │               │
(2, 0): ───H───H───┼───────────────M─────────────
                   │               │
(3, 0): ───H───────┼───@───H───────M─────────────
                   │   │           │
(4, 0): ───H───────┼───┼───@───H───M─────────────
                   │   │   │       │
(5, 0): ───H───H───┼───┼───┼───────M─────────────
                   │   │   │       │
(6, 0): ───H───H───┼───┼───┼───────M─────────────
                   │   │   │       │
(7, 0): ───H───H───┼───┼───┼───────M─────────────
                   │   │   │
(8, 0): ───X───H───X───X───X─────────────────────
```

In [135]:

```python
# print aws sdk circuit
circ = Circuit()
# intialize
init = qubit_count-1
on = 1
for i in range(init):
    circ.h([i])
circ.x([init])
circ.h([init])
# oracle
target = i+1
if secret_bias_bit:
    circ.x([init])
for q in range(qubit_count):
    if secret_factor_bits[q] == on:
        circ.cnot(q,target)
for i in range(init):
    circ.h([i])
```

```
print(circ)
```

```
T  : |0|1|2|3|4|5|

q0 : -H-H---------

q1 : -H---C-H-----
          |
q2 : -H-H-|-------
          |
q3 : -H---|-C-H---
          | |
q4 : -H---|-|-C-H-
          | | |
q5 : -H-H-|-|-|---
          | | |
q6 : -H-H-|-|-|---
          | | |
q7 : -X-H-X-X-X---

T  : |0|1|2|3|4|5|
```

In [136]:

```
def bitstring(bits):
    return ''.join(str(int(b)) for b in bits)
```

In [137]:

```
# Sample from the circuit a couple times
simulator = cirq.Simulator()
result = simulator.run(cirq_circuit, repetitions=circuit_sample_count)
frequencies = result.histogram(key='result', fold_func=bitstring)
print('\nSampled results:\n{}'.format(frequencies))
```

```
Sampled results:
Counter({'01011000': 3})
```

In [138]:

```python
# Check if we actually found the secret value
most_common_bitstring = frequencies.most_common (1) [0] [0]
print('\nMost common matches secret factors:\n{}'.format(
    most_common_bitstring == bitstring(secret_factor_bits)

))
```

```
Most common matches secret factors:
True
```

In [139]:

```python
# Select an S3 bucket to save output.  Change the bucket_name to the S3 bucket you created.
aws_account_id = boto3.client("sts").get_caller_identity()["Account"]
my_bucket = f"amazon-braket-8ec782c949c9" # the name of the bucket
my_prefix = "simulation-output" # the name of the folder in the bucket
s3_folder = (my_bucket, my_prefix)

# Select device arn for the managed simulator
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
```

In [140]:

```python
# run aws device
get_result(device, circ, s3_folder)
```

```
Status: CREATED
Status: QUEUED
Status: RUNNING
Status: COMPLETED
measurement_counts: Counter({'01011001': 504, '01011000': 496})
```

Out[140]:

```
Counter({'01011001': 504, '01011000': 496})
```

In [ ]: