**Лабораторный практикум по дисциплине**

**«Квантовая обработка информации»**

Оглавление

[Введение 191](#_Toc62078002)

[1. Сравнение моделей квантовых вычислителей (мкв) и классификация 192](#_Toc62078003)

[полнофункциональные модели 193](#_Toc62078004)

[надстройка в математической среде 195](#_Toc62078005)

[модель 195](#_Toc62078006)

[контрольные вопросы 208](#_Toc62078007)

[2. Языки программирования для квантовых компьютеров 208](#_Toc62078008)

[2.1. Open quantum assembly language (openqasm) 208](#_Toc62078009)

[2.2. Q# 210](#_Toc62078010)

[2.3. Liqui (language-integrated quantum operations) 211](#_Toc62078011)

[2.4. Quantum computation language (qcl) 211](#_Toc62078012)

[2.5. Quipper 212](#_Toc62078013)

[2.6. Другие подходы 214](#_Toc62078014)

[Контрольные вопросы 215](#_Toc62078015)

[3. Симулятор openqasm 216](#_Toc62078016)

[3.1. Исполнение схемы системы 218](#_Toc62078017)

[3.2. Обзор circuit composer 220](#_Toc62078018)

[3.2.1. Композитор схем 221](#_Toc62078019)

[3.2.2. Визуализируйте состояния кубита 223](#_Toc62078020)

[3.2.3. Создайте свою схему с помощью перетаскивания 224](#_Toc62078021)

[3.3. Модификатор управления 226](#_Toc62078022)

[3.4. Условный сброс 227](#_Toc62078023)

[3.5. Внутренние компоненты sdk: компиляция схемы и qasm 229](#_Toc62078024)

[3.5.1. Компиляция схемы 230](#_Toc62078025)

[3.5.2. Результаты выполнения 232](#_Toc62078026)

[3.6. Графический редактор 233](#_Toc62078027)

[3.7. Пример теста bell 234](#_Toc62078028)

[3.8. Пример состояния ghz 234](#_Toc62078029)

[3.9. Пример состояния w 235](#_Toc62078030)

[3.10. Пример алгоритма гровера 235](#_Toc62078031)

[3.11. Просмотр вероятностей измерения 235](#_Toc62078032)

[3.12. Фаза 236](#_Toc62078033)

[3.13. Пониженная чистота состояния кубита 236](#_Toc62078034)

[3.14. Q-сфера 237](#_Toc62078035)

[3.15. Представление statevector 238](#_Toc62078036)

[3.16. Неунитарные операторы и модификаторы 239](#_Toc62078037)

[3.17. Усиление амплитуды 241](#_Toc62078038)

[3.17.1. Примеры схем 243](#_Toc62078039)

[3.18. Квантовая оценка фазы 244](#_Toc62078040)

[3.18.1. Схема оценки фазы (+) 246](#_Toc62078041)

[3.19. Алгоритм шора 247](#_Toc62078042)

[3.20. Период нахождения 248](#_Toc62078043)

[3.21. От факторинга к поиску периода 248](#_Toc62078044)

[3.22. Контролируемые операции и оценка фаз 249](#_Toc62078045)

[Контрольные вопросы 252](#_Toc62078046)

[4. Описание лабораторных работ 253](#_Toc62078047)

[Лабораторная работа №1. Знакомство со средой моделирования и разработки qasm и ibm quantum experience. Составление модели квантовой системы 253](#_Toc62078048)

[контрольные вопросы 258](#_Toc62078049)

[Лабораторная работа №2. Выполнение эффекта квантовой телепортации в среде моделирования и разработки qasm и ibm quantum experience 259](#_Toc62078050)

[контрольные вопросы 266](#_Toc62078051)

[Лабораторная работа №3. Реализация поискового алгоритма дойча-джозса в среде моделирования и разработки qiskit и ibm quantum experience 267](#_Toc62078052)

[контрольные вопросы 286](#_Toc62078053)

[Лабораторная работа №4. Реализация поискового алгоритма гровера в среде моделирования и разработки qiskit и ibm quantum experience 287](#_Toc62078054)

[контрольные вопросы 308](#_Toc62078055)

# ВВЕДЕНИЕ

Квантовый компьютер – вычислительное устройство, использующее для вычислений явления квантовой механики. Принципиальное отличие от классических компьютеров, состоит в том, что квантовые алгоритмы работают с вероятностными величинами и используют в вычисления квантовый параллелизм, суперпозицию состояний квантовых частиц и квантовую запутанность, что позволяет производить вычисления на других принципах и решать новые классы задач за обозримое время.

На данный момент существует большое количество различных сред моделирования (как консольных, так и графических), библиотек API, а также моделей отдельных алгоритмов. Графические среды моделирования предоставляют большие возможности для построения, и управления процессом моделирования. Среди всех графических сред jQuantum, QCAD ориентированы на построение и моделирование алгоритмов с помощью квантовых схем, но они предоставляют совершенно разные возможности. Bloch Sphere Simulator (BSS) и Squankum позволяют проводить моделирование воздействия операторов на кубит одновременно визуализируя их действие на трехмерной модели кубита – сфере Блоха. Некоторые модели строятся на базе существующих математических сред моделирования, например модель алгоритма Гровера, разработанная в Санкт-Петербургском Государственном Университете. Различаются модели и по подходам к моделированию: QuIDDPro имеет математическое ядро построено на графовом подходе, который в отличии от обычного матричного подхода позволяет значительно экономить память при моделировании. Отдельного упоминания стоят библиотеки APIlibquantum, Cove для построения программ моделирования квантовых вычислений, которые предоставляют готовый функционал для построения собственной модели

При создании модели квантового вычислителя разработчики преследуют совершенно разные цели (моделирование квантовых систем и кубит, воздействия отдельных гейтов или моделирование квантовых алгоритмов) и различные подходы в реализации интерфейса (графический, консольный), видах моделирования. В связи, с чем модели значительно отличаются друг от друга, так же это приводит к тому, что в данной области нет единого направления развития. Вследствие этого, возникает необходимость анализа существующих сред моделирования их классификации, систематизации разрозненных данных о подходах, целях и результатах данных разработок.

# 1. Сравнение моделей квантовых вычислителей (МКВ) и классификация

К настоящему времени разработано большое количество различных программных моделей (как консольных, так и графических), библиотек API, а также моделей отдельных алгоритмов, большинство распространяется свободно и доступно для широкого круга пользователей. Предоставляемые возможности моделирования и подходы к управлению настолько различны, что оценить их качество и выбрать подходящую модель – сложная задача. Для анализа были отобраны различные модели, но наиболее интересными являются модели с графическим интерфейсом.

Рассмотрим наиболее проработанных представителей сред моделирования различных классов, отличающиеся как подходами к процессу моделирования, так и решениями в области пользовательского интерфейса. Рассматривая и анализируя возможности существующих моделей, можно произвести сравнение и суммарную оценку моделей по всем характеристикам. Анализ производится по категориям характеристик, наиболее важные из которых затрагиваются в пунктах «Моделирование» и «Архитектурные и программные характеристики».

Можно выделить три вида моделей/приложений для моделирования квантовых вычислений:

## Полнофункциональные модели

Модели этого вида являются полностью самодостаточными в своем функционировании, позволяют строить и моделировать квантовые алгоритмы. Являются кроссплатформенными, но в большинстве своем направлены на решение задач, которые ставил перед собой разработчик.

**Одноалгоритмные модели.** Выполняется моделирование вычислений одного определенного квантового алгоритма. Например, приложения, моделирующие алгоритмы Шора и Гровера. Таковой вид моделей принято называть симуляторами, так как они направлены на демонстрацию различных возможностей. Интерфейс такой модели чаще всего представляет собой окно консоли, представленной на рис. 1.

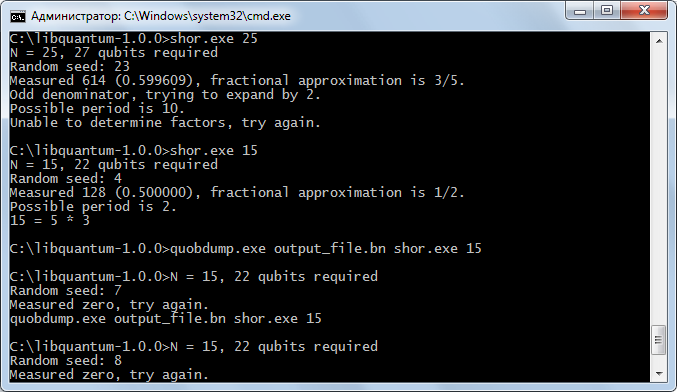


Рис. 1. Запуск и выполнение моделирования алгоритма Шора

**Среда для моделирования.** Включают в себя достаточный набор функциональных возможностей для самостоятельного моделирования квантовых вычислений. К таковым относятся jQuantum, SimQubit, QCE, интерфейс которой представлен на рис. 2, LinearAL, QuIDDPro и т.д.



Рис. 2. Интерфейс QCE

В данных моделях набор операторов сформирован в виде библиотек, которые доступны для редактирования. Для каждой отдельной программы создается отдельный набор квантовых операторов, который необходимо подгрузить в среду перед началом моделирования. Имеющиеся в среде наборы позволяют моделировать два вида среды: идеальную среду квантового вычислителя и среду квантового вычислителя на основе эффекта ядерного магнитного резонанса. Алгоритмы квантовых вычислений программируются в виде последовательности операторов, которые затем последовательно выполняются. Выполнение может производиться в пошаговом и автоматическом режиме.

## Надстройка в математической среде

Данные модели могут представлять собой различного рода компоненты (библиотеки, файлы и т.д.), которые можно связать с такими средами, как MatLab, Maple, Mathcad, данные модели дополняют возможности основной среды для легкого построения моделей квантовых алгоритмов.

**Библиотека, Toolkit.** Содержит все необходимые функции для написания приложения, моделирующего квантовые вычисления. В качестве примера можно привести Toolkit для Wolfram Mathematica, toolbox для MatLab.

## Модель

Модель, отражающая процесс вычисления отдельного алгоритма или квантовый процесс. Например, Маплет для Maple. Данный симулятор представляет собой маплет (код для Maple), иллюстрирующий поведение квантового алгоритма Гровера, поэтому тут есть некоторая свобода в возможностях управления процессом моделирования, если у пользователя есть навыки программирования в Maple. После запуска кода, появляется окно управления моделированием.

Маплет производит вычисления и строит график вероятности для каждого шага вычислений, оптимальное количество шагов () отображается после вычислений. Вышеизложенные виды моделей разделяются по функциональному и интерфейсному принципам, но я считаю, что стоит также выделить модульность, одну из важнейших квантовых характеристик, как еще один из признаков классификации моделей. В данном случае речь идет об архитектурной / программной характеристике. Можно выделить многомодульные и одномодульные модели.



**Одномодульные модели.** Модели, состоящие из одного файла (модуля), не имеют никаких зависимостей, но и не предоставляют возможностей изменений в архитектуре модели или её функциональности без перекомпиляции всего проекта. Все подсистемы: ввода-вывода данных, система визуализация и система моделирования – содержаться в одном исполняемом файле. Примерами таких моделей являются jQuantum, QCAD, QuIDDPro.

**Многомодульные модели.** Модель организована таким образом, что отдельные компоненты среды выносятся в различные файлы (компоненты ввода-вывода данных, математическое ядро, наборы гейтов). Это могут быть динамические библиотеки (содержащие в себе компоненты для ввода-вывода данных в модель, моделирующее математическое ядро, плагины для моделирования), так и библиотек, хранящих определенные конфигурации модели, или наборы гейтов.

***Модели, с отделенным математическим ядром.*** Данные модели предполагают разделение приложения минимум на две части: математическое ядро и пользовательский интерфейс. Математическое ядро выносится зачастую в динамическую библиотеку, таким образом, в модели может быть большое количество различных математических ядер, использующих различные подходы при моделировании. В дополнение к этому появляется возможность написать собственное или дополнить, или заменить имеющиеся математические ядра, что дает большую гибкость и потенциально больше возможностей при разработке модели. Руководствуясь этим же принципом, для существующей библиотеки может быть создан абсолютно новый пользовательский интерфейс.

***Модели, с отделенным набором операторов.*** Математическое ядро может быть интегрировано в единый исполняемый файл, но матрицы преобразований хранятся отдельно и могут быть либо заменены, либо дополнены при необходимости. Для этого в среде моделирования должен быть соответствующий компонент или инструкция поясняющая.

Местом хранения операторов преобразования (гейтов) может быть как динамическая библиотека, в которой хранятся базовые матрицы и алгоритмы получения остальных операторов, например, с помощью произведения Кронекера, бинарной операции над матрицами произвольного размера, обозначается ⊗. Результатом является блочная матрица.

***Многомодульная архитектура с системой плагинов.*** Система плагинов подразумевает использование данных из программы в других целях, например, для построения трехмерного визуального представления процесса моделирования, или же плагин может при наличии доступа проводить модификацию данных внутри модели, например, для моделирования процесса декогеренции, при отсутствии такой возможности в модели.

Критерий разбиения «Многопоточность» основывается на том факте, что процесс моделирования квантовых вычислений напрямую связан с обработкой больших матриц (вектор состояний и матрицы операций), а точнее с операцией перемножения матриц. Эти операции достаточно легко распараллеливаются, что даст заметное ускорение при моделировании на многопроцессорных системах.

**Однопоточные модели.** Такие модели не ориентированы на использование в многопроцессорных системах, моделирование выполняется в одном потоке последовательно. При таком подходе нагружается только на одно ядро или процессор, что нерентабельно на современных компьютерах.

**Многопоточные модели.** Модель, разработанная для многопроцессорных систем, позволяет моделировать процесс эволюции квантовой системы быстрее, вычисляя преобразование матрицы в нескольких потоках, не зависящих друг от друга.

***Многопроцессорные и многоядерные системы.*** Многопоточность приложений может быть реализована за счет создания нескольких потоков на разных ядрах одного или нескольких процессоров. Для быстрого создания таких приложений существует API OpenMP, которое поддерживается GCC с версии 3.5 и MS Visual C++ с версии 8.0.

***Многопоточность на уровне потоковых процессоров графических карт.*** Представителями технологий переноса расчетов с центрального процессора на графический являются CUDA от NVidia и ATI Stream Technology (ранее известная как ATI FireStream и AMD Stream Processor) от AMD. Данные технологии разрабатывались разными компаниями специализированно под графические платы своего производства, поэтому дают максимальный прирост производительности в многопоточных расчётах. Так же компании NVidia и AMD производят специализированные вычислители, основанные на серийных графических процессорах, для программирования которых используется вышеперечисленноеAPI.

Технология OpenCL разрабатывается совместно такими компаниями как AMD, NVidia, Intel и другими, её спецификация подразумевает независимость от аппаратной реализации, что приносит универсальность приложениям, использующих данную технологию вычислений на графических ядрах. Увеличение производительности здесь меньше, по сравнению со специализированными технологиями.

Для реализации преобразований в квантовых вычислениях используются матрицы, размер которых экспоненциально зависит от количества моделируемых кубит, объем памяти одна из главных проблем моделирования квантовых вычислений.

**Неоптимизированное потребление.** Во многих сред моделирования и библиотеках никак не оптимизируют хранение больших матриц, используются полноразмерные матрицы вектора состояний и матрицы преобразований (гейтов). Преимуществом в данном подходе будет возможность задать и хранить любую матрицу, как в SimQubit, и использовать её во время вычислений.

**Оптимизированное.** Больше всего памяти потребляют квадратные матрицы преобразований (NxN), так как их размерность N растет экспоненциально: , где n – количество моделируемых кубит. Матрицы состояний потребляют меньше, так как они фактически одномерные (1xN), под каждое состояние выделяется память для хранения амплитуды этого состояния.

Полноразмерные матрицы операций хранить совсем не обязательно, их можно генерировать во время выполнения, но еще более эффективным способом снизить затраты будет не генерация подобной матрицы, последовательное применение небольших матриц, суммарное воздействие которых будет аналогичным применению одной большой матрицы.

Существует еще один способ уменьшения потребляемой памяти – это оптимизировать хранение матриц состояний. Разработчики Libquantum реализовали вычисления таким образом, чтобы хранить значения амплитуды состояний, которых не равнялась нулю, остальные добавлялись по мере появления в них амплитуды. Разработчики QuIDDPro пошли еще дальше и реализовали вычисления с помощью графов. Во время процесса моделирования рассматриваются величины амплитуд и преобразования этих амплитуд, определяя, таким образом, наиболее вероятные состояния, величины которых могут быть далее использования, остальные состояния просто отбрасываются.

Кроссплатформенность. Данный критерий подразумевает разбиение моделей на группы по степени независимости от платформы (операционной системы и аппаратуры), на которых возможен запуск данной модели.

**Модель, ориентированная для одной платформы.** Модель собирается только для одной операционной системы и процессорной архитектуры, и может быть использована только на ней, многие модели разрабатываются только под платформу UNIX\Linux или Windows, что не очень удобно.

**Модель, ориентированная для нескольких платформ.** Среда моделирования предоставляется в нескольких вариантах, позволяющих использовать её на различных платформах. Модели, разработанные для нескольких платформ, частично облегчают ситуацию, но кроссплатформенная модель наиболее предпочтительна.

Существует достаточно большое количество характеристик, которыми обладают те или иные модели квантового компьютера. Некоторые из них, конечно, имеют похожую функциональную направленность, но все же отличаются по своему содержанию и конкретному предназначению. Постараемся произвести подобную группировку характеристик, так как это поможет не только лучше понять устройство работы моделей квантовых вычислителей, но и усовершенствовать уже имеющиеся модели. Для начала перечислим широко известные характеристики:

1. Модульность (100)
2. Библиотечный набор гейтов (67)
3. Исходное количество гейтов (-)
4. Возможность добавления гейтов (90)
5. Многопоточное моделирование (93)
6. Редуцированная матрица состояний (69)
7. Поддержка платформы Windows (88)
8. Поддержка платформы Linux (87)
9. Кроссплатформенность (98)
10. Редактор квантовой схемы (77)
11. Редактор программного кода (77)
12. Задание входных значение кубит (83)
13. Числовой вывод вероятностей/амплитуд кубита (74)
14. Числовой вывод амплитуд состояний всей группы кубит (75)
15. Числовой вывод амплитуд заданной группы кубит (78)
16. График вероятностей/амплитуд состояний кубит (63)
17. Таблица цветов вероятностей/амплитуд состояний кубит (51)
18. Трехмерная модель кубита (47)
19. Автоматический режим (37)
20. Пошаговый режим (40)
21. Отображение текущих результатов (55)
22. Вывод матрицы квантовогогейта (66)
23. Моделирование физических процессов (96)
24. Матричное математическое ядро (60)
25. Графовое математическое ядро (60)
26. Нет ограничения на количество кубит (95)
27. Динамическое добавление кубит (43)
28. Сохранение/загрузка программы/схемы/алгоритма (30)
29. Сохранение результатов вычислений (32)
30. Загрузка и продолжение вычислений (36)
31. Неограниченная величина алгоритма/схемы/программы (43)
32. Открытая архитектура (94)
33. Отработанные примеры (28)
34. Документация (27)

Многие из этих характеристик встречаются и в классических компьютерах, но появляются и новые, ранее неизвестные для простого человека понятия.

**Модульность**

В данном случае под модульностью подразумевается разделение среды моделирования на отдельные файлы-компоненты, которые содержат в себе различные части модели. Это могут быть: исполняемые файлы, библиотеки, файлы баз данных и т.д. Программный код самой модели зачастую разделяется на несколько частей-модулей, но при компиляции собирается один исполняемый файл. Можно выделить многомодульные и одномодульные модели. Этой характеристикой обладает достаточно ограниченное число моделей квантовых вычислителей.

**Поддержка платформы Windows**

Платформа Windows – это семейство проприетарных операционных систем корпорации Microsoft, ориентированных на применение графического интерфейса. По состоянию на сентябрь 2014 года под управлением операционных систем семейства Windows по данным ресурса NetMarketShare (NetApplications) работает около 90% персональных компьютеров. Это свидетельствует о значимости наличия данной характеристики для моделей квантовых вычислителей. Ведь персональные компьютеры являются важной частью “мировой техники”.

**Поддержка платформы Linux**

Linux – общее название Unix-подобных операционных систем, основанных на одноимённом ядре. Оно создаётся и распространяется в соответствии с моделью разработки свободного и открытого программного обеспечения. Данная методология очень близка к принципам, ожидаемым от оптимальной модели квантового вычислителя, так как основой всего является модульность, к чему и следует стремиться при создании моделей, имеющих свой набор признаков и особенностей или же уже настроенных под конкретные нужды пользователя. В настоящее время системы Linux лидируют на различных рынках техники, но наиболее важным является лидерство на рынке самых мощных суперкомпьютеров (97%), так как именно на такого рода технике и возможно сейчас полное, с точки зрения свойств и функциональности, моделирование и работа реальных квантовых вычислителей, не говоря уже об их прототипах и моделях.

**Кроссплатформенность**

Под данной характеристикой подразумевается независимости модели от платформы (операционной системы и аппаратуры), на которых возможен запуск и последующая работа данной модели. Модели могут собираться только для одной операционной системы и процессорной архитектуры, и могут быть использованы только на ней. Могут быть разработаны для нескольких платформ. Или могут быть спроектированы как кроссплатформенное приложение. Это достигается реализацией на языках, не зависящих от среды выполнения, таких как Java или .Net.

**Матричное математическое ядро**

Математическое ядро в виде матрицы. Матричный подход предоставляет точные результаты, но потребляют очень большое количество ресурсов. Различие подходов к типу ядра влияет на точность и достоверность результатов моделирования. Таким ядром наделены все рассматривающиеся в работе модели, кроме QuIDDPro.

**Графовое математическое ядро**

Моделирование с помощью переходов по графам позволяет значительно экономить память и время необходимое на моделирование, но достоверность предоставляемых результатов оставляет желать лучшего.

**Открытая архитектура**

Для построения модели, которая предполагает возможность модификации модели и дополнения новыми функциями, при разработке необходимо придерживаться нескольких принципов. Разделение модели на несколько модулей выполняющих строго определенные спецификацией функций позволяет построить модель, компоненты которой могут быть легко заменены, а открытые исходные коды и открытая спецификация способствуют продвижению и дополнению программного продукта другими разработчиками, что реализовано в таких моделях, как jaQuzzi, QCAD, jQuantum.

**Библиотечный набор гейтов**

Подобно классическому компьютеру, модель квантового вычислителя может быть построена из схем логических квантовых гейтов. Известно, что при помощи классических логических гейтов NOT, AND и OR можно построить схему для вычисления любой булевой функции, так как данный набор гейтов составляет полный базис. По тем же принципам и практически тем же, по своим функциям, гейтам можно построить любую квантовую схему. Выделяют одно-(NOT), двух-(CNOT или XOR), трехкубитовые (CCNOT) и универсальные гейты (Фредкина, Тоффоли). Также можно использовать гейт контролируемой унитарной операции, представленный на рис. 3.

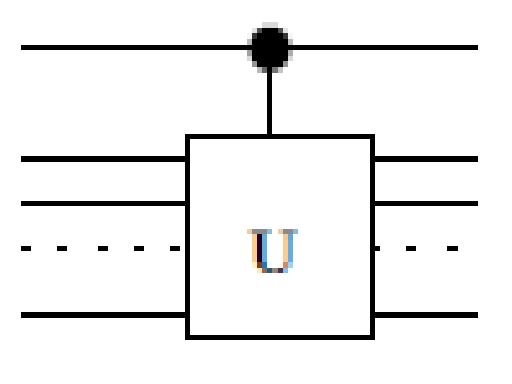


Рис. 3. Контролируемый унитарный гейт

Стоит сделать вывод, что наличие в модели всех вышеперечисленный гейтов не обязательно, так как всего несколько из них уже образуют полный базис. Но чем их больше, тем квантовая схема будет менее запутанной как для создания, так и для ее дальнейшего понимания и перепостроения.

**Исходное количество гейтов**

Подразумевается строгое ограничение на количество гейтов, которое будет предварительно вычислено, а затем они будут выполнены на квантовом устройстве за одну пакетную операцию с возможным измерением всех кубитов в конце. Выражается во всех моделях в разном числовом значении.

**Возможность добавления гейтов**

Существует одна операция, которая доступна в моделях – это возможность добавлять и изменять последовательность квантовых гейтов в ответ на результаты предыдущих измерений. Такая особенность бывает полезна в случаях, когда, например, неверно произведена оценка количества гейт.

**Редуцированная матрица состояний**

Важным использованием редуцированных матриц плотности (состояний) является их возможность описывать состояния подсистем комплексных (составных) квантовых систем. Это полезно, так как при помощи векторов стояний это сделать невозможно.

**Редактор квантовой схемы**

Квантовая схема – это последовательность физических преобразований из конечного набора базисных элементарных преобразований – гейтов. На вход квантовая схема получает квантовые биты. Результат ее работы вероятностный. Редактор позволяет производить различного рода манипуляции с элементами квантовой схемы и т.д.

**Редактор программного кода**

На первый взгляд можно подумать, что такая характеристика, как наличие редактора кода сегодня не вызывает вопроса: “Что это такое?”. Но в данном случае более интересным является тот факт, что он содержит два режима работы:

**Интерактивный режим**

В модели реализован интерфейс диалога человека и машины, пользователь с помощью клавиатуры вводит команды, а модель их тут же выполняет и отображает результат. Такой режим работы поддерживают LinerAl, PyQu. В интерпретаторе гейты хранятся в виде готовых функций, для ознакомления с которыми можно воспользоваться подробным описанием на английском языке. Функции для использования в процессе вычислений можно писать самому. В среду уже строены программы обучения работы с продуктом и простым квантовым вычислениям.

**Пакетный режим**

Позволяет пользователю на входе задать входные данные и последовательность команд, выполнение которых происходит в модели, а на выходе пользователь получает результат вычислений.

**Автоматический режим**

Автоматический режим предполагает, что большинство действий при работе модели квантового вычислителя берет на себя сама программа. При возникновении важных событий модель автоматически выполняет действие, рекомендуемое (заложенное в модель) ее разработчиками. При обнаружении ошибки модель пытается ее исправить или предложить пользователю на выбор несколько дальнейших исходов работы.

**Пошаговый режим**

Вся работа (процессы) модели квантового вычислителя зависит от постоянного контроля пользователя и выбора определенных действий.

**Многопоточное моделирование**

Процесс моделирования квантовых вычислений напрямую связан с обработкой больших матриц (вектор состояний и матрицы операций), а точнее с операцией перемножения матриц. Эти операции достаточно легко распараллеливаются, что даст заметное ускорение при моделировании на многопроцессорных системах.

**Задание входных значений кубит**

Организация системы ввода вывода имеет большое значение для удобства работы пользователя, чем меньше пользователь тратить время на перевод выданной информации в нужный ему вид, тем более продуктивный будет его труд, пользователь меньше отвлекается от задачи и более сосредоточен на ней. Подобная характеристика есть практически во всех описанных в работе моделях.

**Вывод матрицы квантового гейта**

Квантовый гейт осуществим, если и только если соответствующая матрица унитарна, другими словами, если обратная есть сопряженная и транспонированная. Матрица (система) квантовых гейтов представляет собой массив квантовых гейтов и логических проводов, соединяющих их входы и выходы. Входные данные для массива гейтов прогоняются через последовательность квантовых гейтов. Значения битов определяются после действия последнего гейта, и полученные значения будут выходными данными. Эта модель аналогична классической ациклической схеме в теории вычислений. Сравнивая массивы гейтов с квантовой машиной Тьюринга, нам надо ввести условия, которые выделяют массивы гейтов в универсальные классы сложности.

**Нет ограничения на количество кубит**

В некоторых моделях ограничен объем моделируемой системы кубит определенным значением, а остальные только возможностями компьютера. Необычное ограничение в величине моделируемого количества шагов представлено, например, в QCAD.

**Динамическое добавление кубит**

Иногда в процессе моделирования возникает потребность динамического добавления кубит во время моделирования, это реализовано далеко не во всех моделях, а именно в Libquantum, Cove.

**Сохранение/загрузка программы/схемы/алгоритма**

Данная характеристика предполагает наличие у модели квантового вычислителя возможностей сохранения программы, квантовой схемы или алгоритма для выполнения вычислений.

**Сохранение результатов вычислений**

Пользователь не всегда заранее верно может определить время вычислений или работы того или иного процесса. Поэтому возможность приостановки и сохранения результатов моделирования очень удобна для продолжения моделирования в другое время или построения различных вариантов последующего моделирования и последующего анализа.

**Загрузка и продолжение вычислений**

Предполагается возможность загрузки в любое время и непосредственное продолжение работы модели с места сохранения.

**Неограниченная величина алгоритма/схемы/программы**

Определяется неограниченностью:

* в объеме по написанию алгоритма/программа;
* в количестве элементов квантовой схемы.

**Числовой вывод вероятностей/амплитуд кубита**

Общеизвестный факт, что состояние кубита описывается как суперпозиция состояний |0> + |1>. При этом коэффициенты и – это комплексные числа, называемые амплитудой вероятности обнаружить кубит в соответствующем состоянии при измерении. Числовые значения данных величин необходимы для получения точных значений результатов моделирования, это может быть полезно для тестирования модели и проверки правильности её работы.

**Числовой вывод амплитуд состояний всей группы кубит**

Данная характеристика отвечает за возможность отображения числовых значений амплитуд состояний всех используемых кубит.

**Числовой вывод амплитуд заданной группы кубит**

Иногда требуется ограничить числовой вывод амплитуд заданной группы кубит для рассмотрения какого-либо кубита в отдельности или их совокупности.

**График вероятностей/амплитуд состояний кубит**

В большинстве моделей, имеющих данную характеристику, график представляет собой не что иное, как гистограмму вероятностей/амплитуд состояний кубит.

**Таблица цветов вероятностей/амплитуд состояний кубит**

Вывод информации о состоянии кубит системы с помощью цветов предоставляет более наглядные данные, которые проще воспринимать, чем числа, но для полноты картины этого иногда бывает недостаточно.

**Трехмерная модель кубита**

Существуют модели, в которых отсутствует основной функционал моделирующей среды. Пример такой модели представлен на рисунке 6. Возникает вопрос: “для чего тогда нужны такие модели?”. Они предназначены для отображения 3-х мерной модели состояния квантового бита и то, как меняются значения параметров кубит при действии различных операторов и квантовых эффектов.

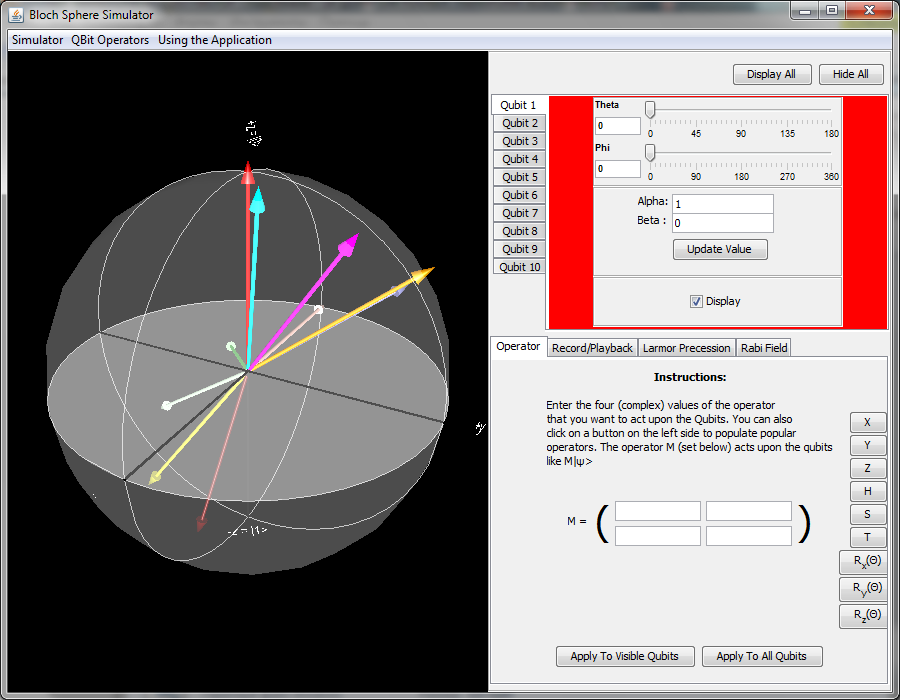


Рис. 4. Интерфейс Bloch Sphere Simulator

Среда моделирует 3-х мерное визуальное представление состояния кубит на сфере Блоха. Фактически данный продукт не является полноценной средой моделирования, он способен динамически отображать воздействия только гейтов оперирующих над одним кубитом, а также моделирует некоторое время эволюции системы (кубита или их совокупности) под воздействием магнитного поля.

**Отработанные примеры**

Иногда очень удобным способом изучения какой-либо новой для себя программы или приложения является рассмотрение ранее отработанных и занесенных в комплект с приложением примеров работы отдельного процесса или всего приложения в целом. Можно наглядно освоить многие аспекты работы с приложением, начиная с того каких действий ждет приложение от пользователя и заканчивая обработкой и анализом полученных результатов работы самого приложения.

**Документация**

Наличие конструкторской, технической, эксплуатационной и другого рода документации. Документация будет представлять набор документов, используемых при проектировании (конструировании), создании (изготовлении) и использовании (эксплуатации) технического объекта.

Целью выполнения описанных лабораторных работ является исследование существующих моделей квантовых вычислителей, выявление достоинств и недостатков, качественная оценка и создание моделей и алгоритмов квантового вычислителя, опираясь на результаты оценки.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое квантовое вычислительное устройство
2. Перечислите виды моделей/приложений для моделирования квантовых вычислений
3. В чем отличие многопоточных от много процессорных квантовых систем?
4. Перечислите основные характеристики квантовых вычислительных систем и симуляторов
5. В чем отличие графового математического ядра от матричного
6. Расскажите о трехмерной модели кубита

# 2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ ДЛЯ КВАНТОВЫХ КОМПЬЮТЕРОВ

1. 2.1. Open Quantum Assembly Language (OpenQASM)

Исходный код OpenQASM был выпущен как часть программного обеспечения IBM Quantum Information Software Kit (QISKit) для использования с квантовой вычислительной платформой Quantum Experience. OpenQASM имеет общие черты со специализированными языками программирования (такими, как Verilog), используемыми для описания структуры и поведения электронных схем. Программы QASM фактически всегда начинаются одинаково: мы определяем все биты, которые нам понадобятся – как квантовые, так и нормальные. Ниже приведен пример исходного кода OpenQASM. Программа добавляет два четырехбитовых номера.

include "qelib1.inc";

gate majority a,b,c

{

cx c,b;

cx c,a;

ccx a,b,c;

}

gate unmaj a,b,c

{

ccx a,b,c;

cx c,a;

cx a,b;

}

qreg cin[1];

qreg a[4];

qreg b[4];

qreg cout[1];

creg ans[5];

// устанавливаем входящие значения

x a[0]; // a = 0001

x b; // b = 1111

// добавляем a к b, сохраняем результаты в b

majority cin[0],b[0],a[0];

majority a[0],b[1],a[1];

majority a[1],b[2],a[2];

majority a[2],b[3],a[3];

cx a[3],cout[0];

unmaj a[2],b[3],a[3];

unmaj a[1],b[2],a[2];

unmaj a[0],b[1],a[1];

unmaj cin[0],b[0],a[0];

measure b[0] -> ans[0];

measure b[1] -> ans[1];

measure b[2] -> ans[2];

measure b[3] -> ans[3];

measure cout[0] -> ans[4];

1. 2.2. Q#

Высокоуровневый язык программирования Q# нивелирует необходимость иметь глубокие знания в квантовой физике. Для заинтересованных в учебнике по языку дается информация по основным концепциям квантовых вычислений, охватывающая векторную и матричную математику, кубиты, обозначения Дирака, принцип Паули и квантовые схемы.

operation Teleport(msg : Qubit, there : Qubit) : () {

body {

using (register = Qubit[1]) {

let here = register[0];

H(here);

CNOT(here, there);

CNOT(msg, here);

H(msg);

if (M(msg) == One) { Z(there); }

if (M(here) == One) { X(there); }

}

}

}

Q# выглядит не так, как большинство других языков программирования, и несколько похож на C#.

Quantum Development Kit предоставляется бесплатно с подробными инструкциями по его установке и вводным учебным программам. Q# компилируется на квантовом симуляторе Visual Studio, имитируя квантовый процессор на 32 кубита. Симулятор может имитировать до 40 кубитов. Если следовать туториалу от Microsoft, то процесс обучения пойдет от наблюдения запутанных состояний из двух кубитов к моделированию квантовой телепортации.

1. 2.3. LIQUi (Language-Integrated Quantum Operations)

Платформа LIQUi, созданная командой Quantum Architectures and Computation Group в Microsoft Research, включает язык программирования, алгоритмы оптимизации и планирования, а также несколько квантовых симуляторов. LIQUi может использоваться для перевода квантового алгоритма, написанного в виде программы высокого уровня на языке F# из семейства .NET Framework, в низкоуровневые команды для квантового компьютера. LIQUi позволяет моделировать до 30 кубитов на одной машине с 32 ГБ оперативной памяти. Платформу можно использовать для определения, выполнения и отображения в различных графических форматах квантовых схем. С помощью LIQUi можно имитировать простую квантовую телепортацию, алгоритм факторизации Шора, квантовую ассоциативную память, квантовую линейную алгебру.

operation TeleportClassicalMessage(message : Bool) : Bool {

body {

mutable measurement = false;

using (register = Qubit[2]) {

// Запросим несколько кубитов, которые можно использовать для телепортации.

let msg = register[0];

let there = register[1];

// Кодируем сообщение, которое мы хотим отправить.

if (message) { X(msg); }

Teleport(msg, there);

// Проверяем сообщение.

if (M(there) == One) { set measurement = true; }

ResetAll(register);

}

return measurement;

}

}

1. 2.4. Quantum Computation Language (QCL)

QCL, или Quantum Computation Language создан Бернхардом Омером в 1998 году. Развитие языка продолжается и сейчас: существует эмулятор, который позволяет запускать квантовые программы на классическом компьютере. Конечно, эмулятор не может обеспечить ускорение квантового параллелизма; с другой стороны, он предлагает программисту некоторые полезные функции, такие как команды для проверки внутреннего состояния кубитов (что крайне трудно сделать на реальном квантовом оборудовании). QCL заимствует синтаксис C и Java, которые иногда описываются как «императивные» языки, потому что они полагаются на прямые команды для установки и сброса значений переменных. Такие команды обычно запрещены в квантовом вычислении, поэтому основные части программы QCL работают только на классическом оборудовании. Квантовая система служит «оракулом», отвечающим на вопросы, которые могут быть заданы в формате, подходящем для вычислений кубитов. Каждый запрос к оракулу должен иметь требуемую архитектуру дымоходной трубы, но он может быть встроен в цикл во внешнем классическом контексте. Фрагмент кода, созданного в QCL (дискретное преобразование Фурье):

operator dft(qureg q) {

const n=#q;

int i; int j;

for i=1 to n {

for j=1 to i-1 {

V(pi/2^(i-j),q[n-i] & q[n-j]);

}

H(q[n-i]);

}

flip(q);

}

Дискретное преобразование Фурье является решающим шагом в алгоритме факторизации Шора. В алгоритме Шора число, подлежащее факторизации, рассматривается как волнообразный, периодический сигнал. Если N имеет коэффициенты u и v, то N состоит из u повторений v или v повторений u. Алгоритм Шора использует квантовый параллелизм для поиска периода таких повторений, хотя процесс не такой простой и прямой, как может показаться в примере выше.

1. 2.5. Quipper

Язык был создан коллективом авторов под руководством Питера Селингера. Quipper предназначен для тех же задач программирования, что и QCL, но имеет другую структуру и внешний вид. Язык реализован как расширение Haskell, которое использует функциональный, а не императивный способ выражения. Рассмотрим классическую квантовую телепортацию. Она включает в себя две стороны – Алису и Боба. Целью Алисы является телепортация кубита q к Бобу. У Алисы и Боба должен быть доступ к кубитам из запутанной пары (a, b). Мы можем думать о роли Алисы в терминах функции, которая вводит два кубита q и a. На выходе функции будет пара классических бит, созданных Алисой:

alice :: Qubit -> Qubit -> Circ (Bit,Bit)

alice q a = do

a <- qnot a ‘controlled‘ q

q <- hadamard q

(x,y) <- measure (q,a)

return (x,y)

А вот интересный пример возведения в 17 степень, путем возведения x в 16 степень встроенной процедурой возведения в квадрат и перемножением x и x^16:

o4\_POW17 :: QIntTF -> Circ (QIntTF,QIntTF)

o4\_POW17 = box "o4" $ \x -> do

comment\_with\_label "ENTER: o4\_POW17" x "x"

(x, x17) <- with\_computed\_fun x

(\x -> do

(x,x2) <- square x

(x2,x4) <- square x2

(x4,x8) <- square x4

(x8,x16) <- square x8

return (x,x2,x4,x8,x16))

(\(x,x2,x4,x8,x16) -> do

(x,x16,x17) <- o8\_MUL x x16

return ((x,x2,x4,x8,x16),x17))

comment\_with\_label "EXIT: o4\_POW17" (x,x17) ("x","x17")

return (x, x17)

Система Quipper – это компилятор, а не интерпретатор; он переводит полную программу за один раз, а не выполняет инструкции друг за другом. Выход компилятора состоит из квантовых схем: сетей взаимосвязанных, обратимых логических вентилей. Схема может иметь форму электрической схемы, но также представляет собой последовательность инструкций, готовых к выполнению с помощью подходящего квантового оборудования или симулятора. Quipper, как и QCL, автоматически генерирует схемы из высокоуровневых исходных смысловых конструкций.

1. 2.6. Другие подходы

  
Разноцветные квадраты говорят пяти квантовым битам IBM, что нужно делать. Перетаскиванием вы можете создавать свои собственные квантовые вычисления

Проект IBM Quantum Experience предоставляет возможность каждому запустить экспериментальную программу на реальном квантовом компьютере. Работа с языком программирования IBM похожа на процесс написания музыки с помощью приложения. Программист может просто перетащить квантовые объекты в определенную область, чтобы написать программу.

Quantum Computing Playground – эксперимент WebGL Chrome, позволяющий смоделировать работу с квантовым компьютером в окошке браузера. Имеется собственный язык сценариев Qscript с функциями отладки и 3D-квантовой визуализации. Квантовая вычислительная площадка может эффективно имитировать квантовые регистры до 22 кубитов.

Python QISKit SDK включает в себя несколько инструментов, которые инженеры IBM Q предоставили для иллюстрации целей квантового программирования. В частности, SDK показывает, как вы можете выполнить несколько заданий для сложных экспериментов. Как ясно из названия, QISKit позволяет разработчикам исследовать квантовый компьютер с помощью Python.

Qbsolv – open source проект для работы с кубитами квантового процессора D-Wave (подходит только для компьютеров этой компании).

Языков квантового программирования (и симуляторов) уже десятки, но все они работают на виртуальной машине. Вероятно, IBM Q – это единственный проект, который предлагает доступ к реальному квантовому компьютеру. Однако для того, чтобы начать заниматься «квантовым программированием», вовсе не обязательно иметь доступ к реальному передовому устройству. Уже сейчас можно не только изучать работу перспективных квантовых алгоритмов, но и создавать работающие приложения, например игры.

1. КОНТРОЛЬНЫЕ ВОПРОСЫ
2. Описать структуру программ на языке OpenQASM
3. Какие программные и аппаратные ограничения имеет квантовый язык LIQUi (Language-Integrated Quantum Operations)?
4. Какие существуют команды для проверки внутреннего состояния кубитов в QCL?
5. Какой способ выражения имеет язык Quipper? Опишите его особенности
6. Какие функции выполняет компилятор языка Quipper?

# 3. СИМУЛЯТОР OPENQASM

IBM Quantum Experience – это онлайн-платформа, которая предоставляет пользователям в широкой публике доступ к набору прототипов квантовых процессоров IBM через облако, онлайн-форум для обсуждения соответствующих тем квантовых вычислений, набор руководств по программированию устройств IBM Q и других учебных материалов по квантовым вычислениям. Это пример облачных квантовых вычислений. В IBM Quantum Experience три процессора: два 5-кубитных процессора и 16-кубитный процессор. Эту службу можно использовать для запуска алгоритмов и экспериментов, а также для изучения учебных пособий и моделирования возможностей квантовых вычислений. На сайте также представлен легко обнаруживаемый список исследовательских работ, опубликованных с использованием IBM Quantum Experience в качестве платформы для экспериментов.

Квантовые процессоры IBM состоят из сверхпроводящих трансмонов кубитов, расположенных в холодильных камерах в штаб-квартире IBM Research в Исследовательском центре Томаса Дж. Ватсона. Симулятор дает возможность перетаскивать необходимые гейты на ветви квантовой схемы (рис. 5)

###### **_images/composer-visualizations.png**

###### Рис. 5. Программирование с помощью Qiskit в облачной Quantum Lab

Пользователи взаимодействуют с квантовым процессором через модель вычислений квантовой схемы, применяя квантовые вентили к кубитам, используя графический интерфейс, называемый квантовым композитором, записывая код квантового ассемблера или через Qiskit. IBM Quantum Experience позволяет разрабатывать рабочие процессы, включающие как квантовые схемы, так и классические вычисления, и быстро создавать квантовые приложения. Разработка схемы перетаскивания не требует кодирования. Можно создавать квантовые схемы и запускать их на симуляторах или реальном квантовом оборудовании с помощью Circuit Composer. Можно создать, выполнить и анализировать схемы в автономной среде программирования Jupyter.

###### **_images/qiskit.png**

Рис. 6. Доступ к IBM Quantum Experience контролируется

учетной записью пользователя

Эта учетная запись привязана к учетным данным, используемым для доступа к IBM Quantum Experience. Учетная запись пользователя выполняет две важные функции. Во-первых, учетная запись связана с одним или несколькими поставщиками, которые предоставляют доступ к различным услугам, предоставляемым IBM Quantum Experience. Во-вторых, каждой учетной записи назначается уникальный токен, позволяющий получить доступ к IBM Quantum Experience из Qiskit. Вы можете посетить свою учетную запись на этой странице после входа в систему.

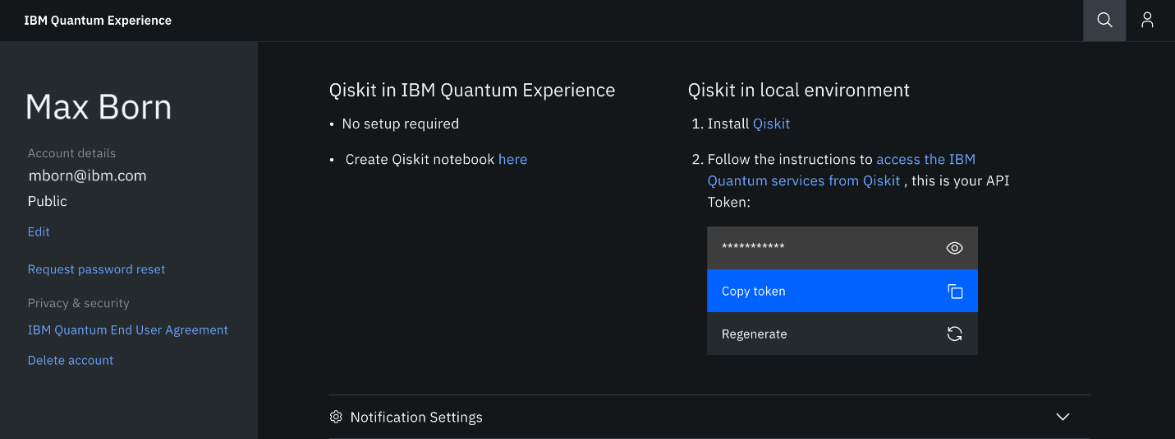
****

Рис. 7. Вход в систему IBM Quantum Experience

IBM Quantum Experience включает высокопроизводительный симулятор для создания прототипов квантовых схем и алгоритмов и изучения их производительности при реалистичных моделях шума устройства. Бэкэнд симулятора называется ibmq\_qasm\_simulator. Базовый код похож на qasm\_simulator, найденный в Qiskit Aer. Ibmq\_qasm\_simulator позволяет выполнять выборку схем до 32 кубитов и поддерживает возможность передачи модели шума для имитации работы схем при работе устройства с шумом. Как показано в информации о конфигурации ниже, имитатор позволяет использовать до 300 цепей на задание и изначально поддерживает широкий спектр унитарных и неунитарных операций.

###### **../../../_images/qasm_sim_modal.png**

Рис. 8. Информация о конфигурации OpenQASM

Обратите внимание, что в отличие от реальных квантовых устройств, где задания помещаются в очередь, IBM Quantum Experience может динамически создавать несколько экземпляров симулятора OpenQASM для удовлетворения спроса; при номинальных рабочих нагрузках никогда не нужно ждать запуска ibmq\_qasm\_simulator. Чтобы симулятор не обрабатывал задания, обработка которых в противном случае не завершилась бы в разумные сроки, задания, отправляемые в ibmq\_qasm\_simulator, ограничены временем выполнения менее 10 000 секунд (~ 2,75 часа).

1. 3.1. Исполнение схемы системы

Большинство систем IBM Quantum выполняют схемы с фиксированной скоростью. Хотя эта частота повторения различается в зависимости от системы, основная модель выполнения такая же и описывается здесь. Во-первых, рассмотрим три схемы, отправленные в систему с использованием отдельных заданий, по одной для каждой цепи. В приведенном ниже примере показано, что происходит с заданиями различной продолжительности. Из-за фиксированной частоты повторения существует переменное время простоя, которое происходит перед запуском схемы, чтобы обеспечить совпадение всей продолжительности с заданной системойчастотой повторения.

Ситуация несколько меняется, когда одни и те же схемы объединяются в одно задание. В этом случае схемы, включенные в задание, выполняются путем повторения схем для каждого запрошенного кадра; выполнение осуществляется по столбцам по матрице схем и снимков.

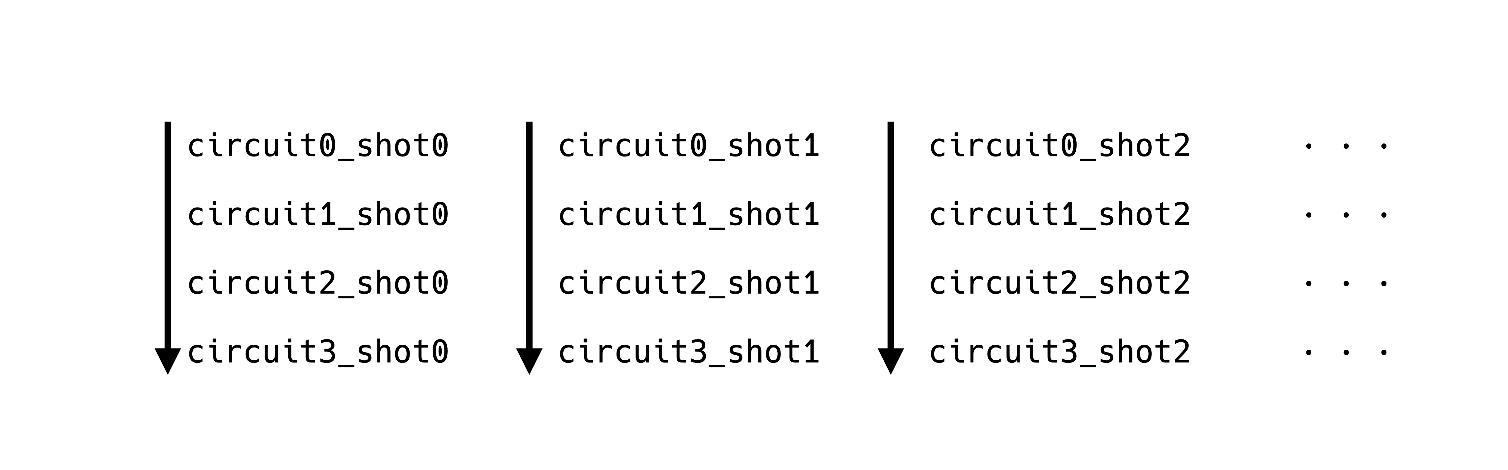


Рис. 10. Повторение схем для каждого запрошенного кадра

Матрица четырех схем в задании, показывающая схему выполнения по схемам. При отправке пакетов схем в системы схемы выполняются иначе, чем если бы они выполнялись по отдельности. А именно, инициализация схемы, т.е. подготовка основного состояния, и измерения (если таковые имеются) выравниваются по всем схемам. Каждая ветвь квантовой схемы может быть измерена при помощи соответствующего элемента, расположенного в ее конце. Количество элементов каждой ветви может иметь прямую связь между элементами своей ветви или соединяться с другими ветвями (рис. 11).

###### ../../_images/fixed_batch_circuit.png

Рис. 11. Квантовая схема IBM Quantum после серии повторений

Следовательно, каждая схема равна по продолжительности самой длинной схеме в пакете, и есть общее время простоя, которое указывается перед всеми схемами, чтобы соответствовать частоте повторения.Некоторые системы IBM Quantum допускают выполнение с динамической частотой повторения. Эти системы идентифицируются в Qiskit с помощью backend.configuration(). Dynamic\_reprate\_enabled и возвращают значение True. В этих системах можно вручную установить указанное выше время простоя, установив rep\_delay отправленного задания. Из приведенных выше рисунков видно, что за счет сокращения времени простоя можно потенциально увидеть большую пропускную способность схем в системах, которые поддерживают динамическую частоту повторения.

1. 3.2. Обзор Circuit Composer

###### Circuit Composer имеет настраиваемый набор инструментов, которые позволяют создавать, визуализировать и запускать квантовые схемы на квантовом оборудовании или симуляторах. Используйте меню «Дополнительные параметры» в каждом окне для доступа к дополнительным инструментам и действиям.

###### Circuit Composer window.

Рис. 12. Основное окно Circuit Composer

1. Строка меню – используйте эти меню для открытия новой цепи, управления и проверки сохраненных схем, настройки рабочего пространства, получения помощи.

2. Область выполнения – измените параметры запуска, а затем запустите схему на квантовом сервере.

3. Файлы Composer – просмотрите и отредактируйте имя схемы или получите доступ к другим файлам Circuit Composer.

4. Редактор кода, панель «Документы» и «Задания» – используйте боковую панель для просмотра редактора кода, документации или списка заданий. Чтобы закрыть боковую панель, нажмите на название открытой вкладки.

• Редактор кода – код OpenQASM и Qiskit (Python) обновляется автоматически по мере построения схемы. Вы можете отредактировать код OpenQASM или открыть код Qiskit в записной книжке в Quantum Lab.

• Документы – быстрый доступ к материалам для начала работы, а также к справочным материалам, практическим рекомендациям и темам обучения.

• Задания – отслеживайте, просматривайте или отменяйте задания.

5. Квантовые вентили и панель операций – это строительные блоки квантовых схем. Перетащите эти вентили и другие операции в графический редактор схем. Ворота разных типов сгруппированы по цвету. Например, классические вентили – синие, фазовые вентили – голубые, а неунитарные операции – серые.

6. Графический редактор схем – здесь вы создаете схему. Перетащите вентили и другие операции на горизонтальные «провода» кубита, составляющие ваш квантовый регистр.

Чтобы удалить затвор из провода, выберите затвор и нажмите X. Чтобы отредактировать параметры и настройки вентилей, которые поддерживают редактирование, выберите вентиль в графическом редакторе и нажмите редактировать (pencil).

7. Боковая панель – перейдите к другим инструментам в IBM Quantum Experience, таким как Dashboard, Quantum Lab.

8. Визуализации. Визуализации характеризуют вашу схему в процессе ее построения. Они используют однократный симулятор вектора состояний, который отличается от серверной части, указанной в параметрах «Выполнить» или кнопке «Выполнить». Обратите внимание, что визуализации игнорируют любые добавленные вами операции измерения. Войдите в систему и нажмите кнопку «Выполнить», чтобы вместо этого получить результаты из указанного бэкэнда.

## 3.2.1. Композитор схем

Перетаскивайте, чтобы создавать, визуализировать и запускать схемы – кодирование не требуется. Circuit Composer – это графический инструмент квантового программирования, который позволяет перетаскивать операции для построения квантовых схем и запускать их на реальном квантовом оборудовании. Создайте свою первую квантовую схему с помощью этого пошагового руководства или перейдите непосредственно в Circuit Composer. Визуализации отображаются в окнах в нижней части рабочего пространства Composer (за исключением фазового диска, который появляется в конце каждого провода кубита). Вы можете выбрать любую комбинацию вектора состояний, вероятностей измерения и визуализаций q-сферы, которые будут отображаться в нижней части рабочего пространства. Выберите или отмените выбор визуализаций в меню «Просмотр».

Загрузите одну из визуализаций в нижней части рабочего пространства Composer, щелкнув меню Дополнительные параметры в окне визуализации. Вы можете загружать визуализации в виде SVG, PNG или CSV базовых данных. Вы также можете загрузить изображения визуализации вероятностей измерений и гистограммы векторов состояний в формате PDF.

###### ../_images/giftour.gif

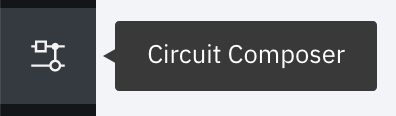
Рис. 13. Средства визуализации Circuit Composer

## 3.2.2. Визуализируйте состояния кубита

Посмотрите, как изменения в вашей схеме влияют на состояние кубитов, показанных в виде интерактивной q-сферы, или гистограмм, показывающих вероятности измерений или моделирование вектора состояний. Запустите свои схемы на реальном квантовом оборудовании, чтобы понять влияние шума устройства. Вместо того, чтобы писать код вручную, автоматически сгенерируйте код OpenQASM или Python, который ведет себя так же, как схема, созданная с помощью Circuit Composer. Создавайте схемы так, как они выглядят в учебниках, и запускайте их на реальном оборудовании, чтобы из первых рук узнать, как они себя ведут. Вы можете использовать Circuit Composer, чтобы построить и запустить полную квантовую схему. Для вашей схемы «Hello world» вы создадите состояние Bell. Визуализации внизу страницы будут автоматически обновляться, чтобы отражать результаты моделирования по мере добавления операций в схему. Однако, если вы хотите запустить свою схему на реальном квантовом оборудовании или на сервере симулятора, или если вы хотите визуализировать более четырех кубитов, вам нужно будет войти в IBM Quantum Experience, нажав кнопку Войти в верхнем правом углу.

*Шаг 1. Композитор открытого цикла.*

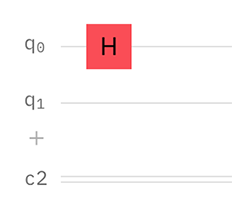
Чтобы открыть Circuit Composer, щелкните значок схемы на левой панели навигации.



Вы увидите набор горизонтальных линий или проводов, которые напоминают лист бумаги в линейку. Верхние провода – это кубиты, а нижний провод – классический регистр.

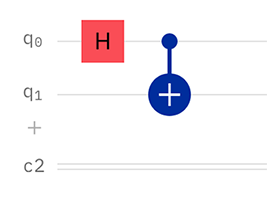
*Шаг 2. Добавьте вентиль (вентиль Адамара) в вашу схему.*

Чтобы добавить вентиль в вашу схему, перетащите операцию из палитры квантовых операций на верхний кубит, q[0].



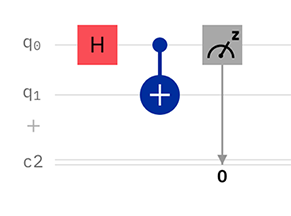
*Шаг 3. Добавьте гейт в вашу схему.*

Чтобы добавить вентиль в вашу схему, перетащите операцию из палитры квантовых операций справа от гейта. Эта операция действует на два кубита.



*Шаг 4. Добавьте операцию измерения.*

Чтобы добавить измерение в вашу схему, перетащите операцию измерения из палитры квантовых операций справа от операции.



Результат измерения записывается классическим битом в классическом регистре. Вертикальный провод, соединяющий операцию измерения с нижним проводом, изображает информацию, текущую от кубита вниз к классическому регистру. Вы построили свою первую квантовую схему. Панели визуализации под вашей схемой дают смоделированный результат, который автоматически обновляется по мере добавления и удаления операций.

## 3.2.3. Создайте свою схему с помощью перетаскивания

###### Перетаскивайте операции из палитры операций в квантовые и классические регистры.

###### После того, как операции размещены в вашей схеме, вы можете продолжать перетаскивать их в новые позиции.

###### Используйте контекстное меню (щелчок правой кнопкой мыши), чтобы скопировать и вставить выбранную операцию. Вы также можете использовать сочетание клавиш (cmd-c или crtl-c, в зависимости от вашей операционной системы), чтобы скопировать выбранную операцию.

###### Если вы используете сочетание клавиш (cmd-v / ctrl-v), операция будет вставлена ​​в тот же регистр, что и скопированная операция.

###### Вы можете выбрать несколько операций, чтобы скопировать и вставить их, или сгруппировать их в настраиваемую унитарную операцию, которая отображается в вашем списке операций и функций как единый вентиль.

###### Чтобы выбрать более одной операции, просто поместите курсор за пределами одной из операций, затем щелкните и перетащите по области, чтобы выбрать. Пунктирная линия показывает, какие операции вы выбираете. Вы также можете щелкнуть отдельные операции, удерживая клавишу Shift, чтобы выбрать или отменить их выбор. Щелкните правой кнопкой мыши внутри области внутри пунктирных линий и выберите в меню «Копировать» или используйте сочетание клавиш cmd-c или crtl-c, чтобы скопировать группу. На рисунке ниже представлена наполненная квантовая схема, в которой элемент |0> отражает начальное квантовое состояние.

###### Right-click menu options.

###### Если для вставки вы используете сочетание клавиш (cmd-v / ctrl-v), группа будет вставлена ​​в те же регистры, что и скопированная группа.

###### Чтобы сгруппировать несколько операций вместе как пользовательскую операцию, сначала сгруппируйте операции, выбрав их, как описано выше, затем щелкните правой кнопкой мыши внутри области внутри пунктирных линий и выберите «Группировать» в меню.

###### Вы также можете использовать сочетание клавиш cmd-g / crtl-g для группировки операций. После группировки операций вам будет предложено назвать настраиваемую операцию, или вы можете принять имя по умолчанию. Нажмите «Создать», и настраиваемая операция будет представлена ​​одним полем как в вашей цепи, так и в списке операций. Теперь вы можете перетащить или скопировать и вставить новую операцию по всей схеме. Обратите внимание, что ваша операция сохраняется в этой цепи, но не будет отображаться в списке операций для других схем. Чтобы разгруппировать ворота в рамках вашей пользовательской операции, щелкните правой кнопкой мыши пользовательскую операцию в Composer и выберите в меню опцию разгруппировать. Вы также можете создать собственную операцию прямо в редакторе кода OpenQASM; чтобы переименовать или удалить пользовательскую операцию, щелкните правой кнопкой мыши операцию в списке операций или в Composer и выберите в меню переименовать или удалить. Обратите внимание, что удаление пользовательской операции из списка операций также приведет к удалению любых ее экземпляров из Composer; однако удаление настраиваемой операции из Composer не удаляет ее из списка операций.

###### Чтобы добавить больше квантовых или классических регистров, перейдите в меню «Правка» и выберите «Управление регистрами». Вы можете увеличить или уменьшить количество кубитов или битов в вашей схеме, а также переименовать регистры. Щелкните «Сохранить», чтобы применить изменения.

1. 3.3. Модификатор управления

###### Модификатор управления дает вентиль, первоначальная работа которого теперь зависит от состояния кубита управления. Перетащите модификатор элемента управления control на ворота, чтобы добавить к нему элемент управления. На кубитах появятся точки, на которые может воздействовать этот элемент управления; щелкните одну или несколько точек, чтобы назначить управление одному или нескольким кубитам. Вы также можете назначить элемент управления, щелкнув ворота правой кнопкой мыши. Чтобы удалить элемент управления, щелкните ворота правой кнопкой мыши и выберите вариант удаления элемента управления.

1. 3.4. Условный сброс

###### Некоторые аппаратные серверные модули теперь могут выполнять условный сброс. Если у вас есть доступ к этим бэкэндам через IBM Q Network, программы для преподавателей или исследователей, теперь вы можете инициализировать (или повторно инициализировать) кубиты в состоянии посредством измерения с последующим условным X-вентилем. Со временем условный сброс будет распространен на большее количество серверных ВМ, включая системы с открытым доступом.

###### • Чтобы открыть редактор кода, щелкните вкладку </> Код.

###### • Вы можете определять свои собственные пользовательские операции;

###### Вы можете определить новые унитарные операции в редакторе кода (см. Пример на рисунке ниже). Операции применяются с использованием имени оператора (params) qargs; точно так же, как встроенные операции. Скобки необязательны, если нет параметров. Чтобы определить пользовательскую операцию, введите ее в редакторе кода OpenQASM в следующем формате: gatename (params) qargs. Если вы нажмете + Добавить в списке операций, вам будет предложено ввести имя для вашей настраиваемой операции, которую вы затем можете создать в редакторе кода. После того, как вы определили свою пользовательскую операцию, перетащите ее в графический редактор и используйте значок редактирования для точной настройки параметров.

###### а) Гейты, которые будут включены в пользовательскую операцию:

###### cust-icon-explod

###### б) Код новой операции:

###### cust-qasm

###### в) Новая операция в графическом редакторе:

###### cust-icon

###### Инспектор схем развенчивает внутреннюю работу создаваемых вами схем. Он обеспечивает настраиваемое пошаговое представление моделирования вашей схемы, так что вы можете видеть состояние кубитов по мере развития вычислений.

###### Circuit Inspector.

###### Щелкните «Просмотр» и выберите окна для визуализаций, которые вы хотите использовать.

###### Щелкните «Проверить» в строке меню. (Обратите внимание, что вы не можете выбрать Inspect, пока не добавите хотя бы одну операцию в Circuit Composer.) Чтобы шаг за шагом перемещаться по визуализациям компонентов вашей схемы, управляйте перемещением Inspector с помощью кнопок вперед и назад.

###### Controls to step forward and backward, and to jump to a circuit's beginning or end.

###### Инспектор управляет по порядку: переход к началу схемы, шаг назад, воспроизведение от одной точки останова к следующей, шаг вперед, переход к концу схемы.

###### С помощью точек останова вы можете определить группы операций для проверки, выбрав одну или несколько операций в вашей схеме, чтобы отметить точку останова. Цветное наложение на операцию указывает, что точка останова установлена. После того, как точки останова определены, вы можете нажать кнопку воспроизведения для последовательного просмотра шагов. Чтобы удалить точку останова, повторно выберите операции, и цветной оверлей исчезнет.

###### Чтобы узнать больше об интерпретации визуализаций.

###### Чтобы закрыть Инспектор и вернуться к редактированию схемы, щелкните значок x в правом верхнем углу панели инспекции.

###### Каждый раз, когда вы открываете Circuit Inspector, симулятор создает случайность, генерируя результаты на основе начального числа. Начальное значение – это начальное значение, введенное в алгоритм, который генерирует псевдослучайные числа. Поскольку семя фиксируется во время сеанса Circuit Inspector, ваши результаты будут повторяемыми. Однако, когда вы закроете Инспектор и снова откроете его, семя будет иметь новое значение, поэтому вы можете увидеть другой результат.

1. 3.5. Внутренние компоненты SDK: компиляция схемы и QASM

На рис. 14 показаны внутренние процессы, происходящие при запуске вашей программы.

* QISKit компилирует схему (схемы) программы в документ в формате JSON, который будет отправлен на локальное моделирующее устройство.
* Управляющее устройство анализирует этот документ, запускает схему и возвращает документ в формате JSON без четкой структуры (скрытый от разработчика).
* QISKit обертывает результирующий JSON-документ в объект, до­ступный для основной программы. Например, вызов result.get\_counts('Circuit') извлекает численные данные из этого документа.

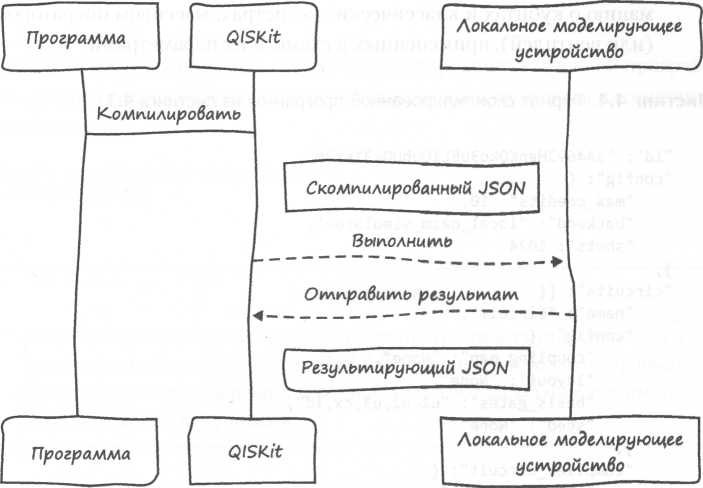


Рис. 14. Схема последовательности действий между программой, QISKit и локальным моделирующим устройством

## 3.5.1. Компиляция схемы

В листинге 1 показан формат скомпилированной программы перед от­правкой на моделирующее устройство. В документе приводятся:

* идентификатор выполнения;
* заголовок с информацией о моделирующем устройстве, включая на­звание, количество кредитов, использованных при выполнении, коли­чество запусков
* раздел схем, содержащий массив объектов. Каждая схема содержит:

1. Название;
2. Заголовок (конфигурацию) с такой информацией, как карта связей кубитов, базовые (физические) вентили, начальное число времени выполнения и т. д.;
3. Раздел скомпилированной схемы с заголовком, содержащим инфор­мацию о кубитах и классических регистрах, массивом операторов (или вентилей), примененных к схеме, и их параметрами.

**Листинг 1.** Формат скомпилированной программы

{

"id": "aA46vJHgnKQko3u5LlQqbUDk31sY2m",

"config": {

"max\_credits": 10,

"backend": "local\_qasm\_simulator",

"shots": 1024

},

"circuits": [{

"name": "Circuit",

"config": {

"coupling\_map": "None",

"layout”: "None", "basis\_gates": "ul,u2,u3,cx,id", "seed": "None"

},

"compiled\_circuit": {

"operations": [{"name": "u3",

"params": [3.141592653589793, 0.0, 3.141592653589793], "texparams": ["\\pi", "0", "\\pi"],

"qubits": [0]

},

{

"name": "measure",

"qubits": [0], "cibits": [0]

}],

"header": {

"number\_of\_qubits": 1, "qubit\_labels": [

["qr", 0]

],

"number\_of\_clbits": 1,

"clbit\_labels”: [

["cr“, 1]

]

}

},

"compiled\_circuit\_qasm": "OPENQASM 2.0;\ninclude Vqelibl.inc \"; \nqreg qr[l];\ncreg cr[l];\nu3(3.14159265358979,0,3.14159265358979) qr[0];\nmeasure qr[0] -> cr[0];\n"

}]

}

Для того чтобы отобразить схему, скомпилированную в вашей програм­ме, распечатайте результат шага компиляции с помощью следующей команды:

qobj = qp.compile(circuits, backend)

print(str(qobj))

## 3.5.2. Результаты выполнения

Это ответный документ от локального моделирующего устройства к QISKit. Формат документа показан в листинге 2. Наиболее значимая информация, содержащаяся в нем:

* Cтатус запуска, время выполнения, название моделирующего устрой­ства и т. д.;
* Результирующие данные. Их можно получить внутри вашей программы с помощью вызова print (str(result.get\_counts('Circuit'))).

**Листинг 2.** Документ с результатами от моделирующего устройства

{

"backend": "local\_qiskit\_simulator", "id": "aA46vJHgnKQko3u5LlQqbUDk31sY2m", "result": [{

"data": {

"counts": {

"1": 1024

},

"time\_taken": 0.0780002

},

"name": "Circuit",

"seed": 123, "shots": 1024, "status": "DONE", "success": true, "threads\_shot": 4

}},

"simulator": "qubit", "status”: "COMPLETED", "success": true,

"time\_taken": 0.0780002

}

Получить документ с результатами немного сложнее, потому что это объект без четкой структуры, недоступный для пользовательской про­граммы. Тем не менее вы можете сохранить скомпилированную схему из предыдущего раздела и вручную передать ее в моделирующее устройство, чтобы получить результат, показанный в листинге 2. Однако это задание оставлено для самостоятельного выполнения. Важно помнить, что до­кумент с результатами, как и формат компиляции, непрозрачен для про­граммиста. Причина в том, что их форматы могут меняться со временем. Тем не менее всегда полезно понимать, что происходит за кадром.

1. 3.6. Графический редактор

###### Используйте графический редактор для создания цепей путем перетаскивания операций. Узнайте больше о том, как создавать схемы с помощью перетаскивания в разделе Создание схемы с помощью перетаскивания.

###### Если вы вошли в систему, вы можете щелкнуть значок файла слева от имени цепи, чтобы получить доступ к списку сохраненных цепей. Этот список состоит из четырех столбцов:

###### Имя (вы назначаете имя при сохранении схемы)

###### Описание (необязательно)

###### Обновлено (дата и время последнего сохранения схемы)

###### ID (идентификационный номер, присвоенный системой)

###### Щелкните имя схемы, чтобы открыть эту схему в Circuit Composer. Установите флажок рядом с именем цепи, чтобы переименовать или удалить цепь, или экспортировать файл OpenQASM. Вы можете удалить или экспортировать несколько цепей одновременно.

###### List of saved circuits, showing a selected circuit and the Export OpenQASM / Rename / Delete buttonsПопробуйте несколько примеров схем

###### • Пример теста Bell

###### • Пример состояния GHZ

###### • Пример состояния W

###### • Пример алгоритма Гровера

###### Нажмите кнопку под изображениями схем, чтобы открыть их в Circuit Composer, где вы можете запустить их на симуляторах или реальном квантовом оборудовании и наблюдать за результатами. Вы также можете редактировать схемы и видеть, как отличается результат.

1. 3.7. Пример теста Bell

###### Тест Белла демонстрирует, что измерения запутанного состояния не могут быть объяснены какой-либо теорией локальных скрытых переменных, и что должны быть корреляции, выходящие за рамки классических. Элементы в конце ветвей q0 и q1 инициируют соответствующее состояние ветви с2.

###### A circuit file for a Bell state.

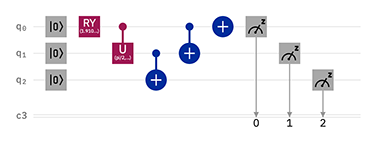
1. 3.8. Пример состояния GHZ

###### Эта схема создает состояние GHZ, а затем измеряет все кубиты в стандартном базисе. Измеренные результаты должны быть половинными.

###### A GHZ state.

1. 3.9. Пример состояния W

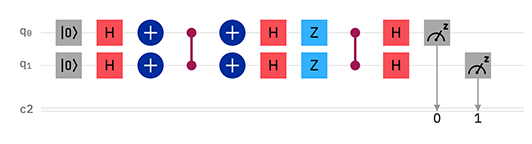
Состояния W – это еще один способ запутать три кубита, и они делают это иначе, чем состояния GHZ.



Прочтите, почему считается, что состояния W имеют более устойчивую запутанность, чем состояния GHZ, в теме с состояниями W и состояниями GHZ.

1. 3.10. Пример алгоритма Гровера

Вот полная схема алгоритма Гровера для этого случая:



1. 3.11. Просмотр вероятностей измерения

В этом представлении вероятности измерения отображаются в виде гистограммы. Вертикальная ось отмечает вычислительные базисные состояния. Метки данных справа от горизонтальных полос обозначают вероятности измерения каждого вычислительного базового состояния в процентах. Рассмотрим следующую квантовую схему и ее вид вероятностей измерения.

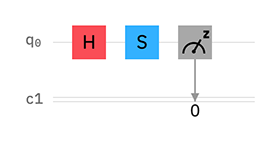
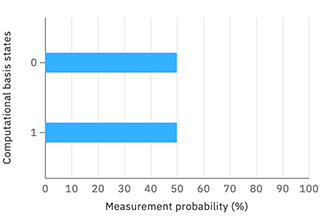
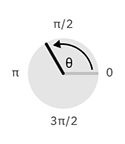
 

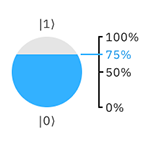
Рис. 15. Квантовая схема и ее вид вероятностей измерения

1. 3.12. Фаза

Фаза вектора состояния кубита в комплексной плоскости задается линией, идущей от центра диаграммы до края серого диска (который вращается против часовой стрелки вокруг центральной точки).

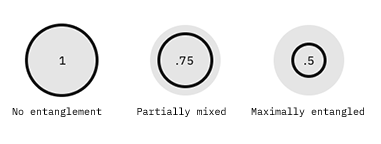


Вероятность того, что кубит находится в состоянии |1> представлена ​​синей заливкой диска.



1. 3.13. Пониженная чистота состояния кубита

Пониженная чистота для одного кубита находится в диапазоне; значение единицы указывает, что кубит не связан с какой-либо другой стороной. Напротив, пониженная чистота показывает, что кубит остается в полностью смешанном состоянии и имеет некоторый уровень сцепления с остальными кубитами и, возможно, даже с окружающей средой.



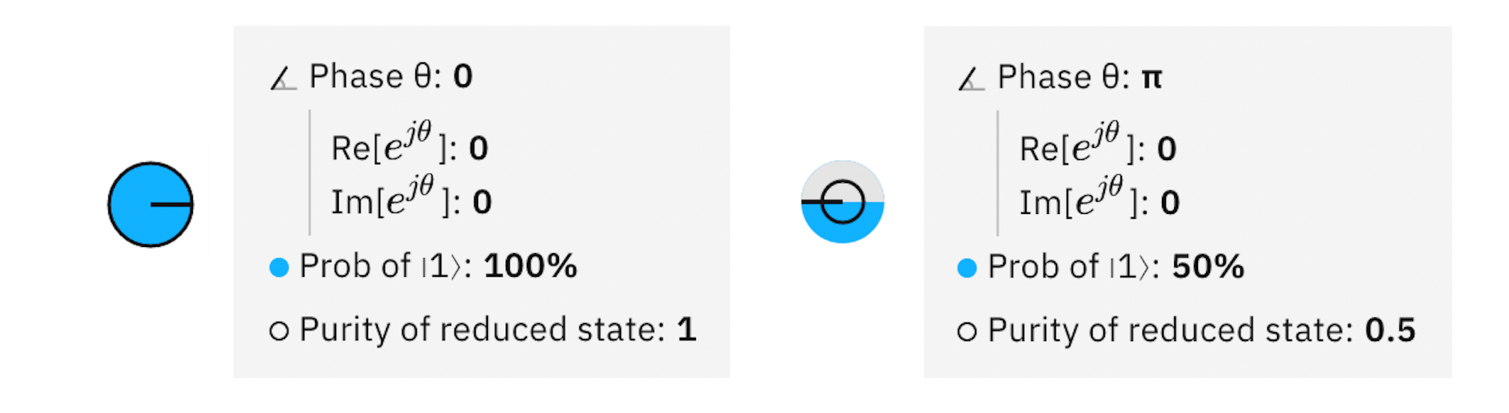


Рис. 16. Два примера визуализации фазового диска

1. 3.14. Q-сфера

Q-сфера представляет состояние системы из одного или нескольких кубитов, связывая каждое вычислительное базовое состояние с точкой на поверхности сферы. В каждой точке виден узел. Радиус каждого узла пропорционален величине комплексной амплитуды его базового состояния, тогда как цвет узла указывает фазу комплексной амплитуды. Узлы располагаются на q-сфере так, чтобы базовое состояние со всеми нулями (например, находилось на его северном полюсе, а базовое состояние со всеми единицами (например,) находилось на его южном полюсе. Базовые состояния с тем же номером нулей (или единиц) лежат на общей широте q-сферы (например,). Начиная с северного полюса q-сферы и продвигаясь к югу, каждая последующая широта имеет базовые состояния с большим числом единиц; широта базисного состояния определяется его расстоянием Хэмминга от нулевого состояния. Q-сфера содержит точно такое же информационное содержание, что и вектор состояний, но предлагает более компактное представление. Рассмотрим следующую квантовую схему и ее q-сферу:

###### a quantum circuit

###### q-sphere view

###### Рис. 17. Квантовая схема и q-сфера, представляющие состояние, создаваемое схемой

###### Квантовая схема показывает направление вектора в каждой из ветвей. Вы можете выбрать, удерживать и перетащить, чтобы повернуть q-сферу. Чтобы вернуть q-сферу в ее ориентацию по умолчанию, нажмите кнопку со стрелкой перемотки вверху справа от q-сферы.

1. 3.15. Представление Statevector

###### Это представление визуализирует вектор состояния в виде гистограммы. Горизонтальная ось обозначает вычислительные базисные состояния. По вертикальной оси измеряется величина амплитуд, связанных с каждым вычислительным базисным состоянием. Цвет каждой полосы представляет фазовый угол каждой амплитуды.

###### Рассмотрим следующую квантовую схему и ее представление вектора состояний:

###### a quantum circuitstatevector view

###### phase colormap

###### Цветовой круг отображает фазовый угол в цвет. Состояние вывода выражается в виде списка комплексных чисел. Схема переводит кубит в состояние. Вычислительные базисные состояния: и. По величине амплитуды обе. Фазы амплитуд равны и для и соответственно. Нажмите на квантовую операцию ниже, чтобы просмотреть ее определение. Операции, больше не используемые в Circuit Composer, перечислены в разделе «Устаревшие операции» в качестве исторической справки.

1. 3.16. Неунитарные операторы и модификаторы

***Сбросить операцию***

Операция сброса возвращает кубит в состояние, независимо от его состояния до применения операции. Это необратимая операция.

reset q [0];



***Измерение***

Измерение в стандартном базисе, также известном как базис z или вычислительный базис. Может использоваться для выполнения любых измерений в сочетании с воротами. Это необратимая операция.

 measure q [0];

***Модификатор управления***

Модификатор управления дает вентиль, первоначальная работа которого теперь зависит от состояния кубита управления. Когда управление находится в состоянии, целевой кубит(ы) претерпевает указанную унитарную эволюцию. Напротив, никакая операция не выполняется, если элемент управления находится в состоянии. Если управление находится в состоянии суперпозиции, то результирующая операция следует из линейности. Перетащите модификатор элемента управления на ворота, чтобы добавить к нему элемент управления. Над и под воротами на проводах кубита появятся точки, которые могут быть контролируемыми целями; щелкните одну или несколько точек, чтобы назначить цель одному или нескольким кубитам. Вы также можете назначить элемент управления, щелкнув ворота правой кнопкой мыши. Чтобы удалить элемент управления, щелкните ворота правой кнопкой мыши и выберите вариант удаления элемента управления.

###### ***IF операция***

###### Операция IF позволяет условно применять квантовые вентили в зависимости от состояния классического регистра.

###### if-oper if (c == 0) x q[0];

###### ***Барьерная операция***

###### Чтобы сделать вашу квантовую программу более эффективной, компилятор попытается объединить вентили. Барьер – это инструкция компилятору, чтобы предотвратить эти комбинации. Кроме того, это полезно для визуализаций.

###### barrier barrier q;

1. 3.17. Усиление амплитуды

###### Прежде чем взглянуть на список предметов, мы понятия не имеем, где находится отмеченный предмет. Следовательно, любое предположение о его местонахождении ничуть не хуже любого другого, которое может быть выражено в терминах квантового состояния, называемого однородной суперпозицией.

###### Если бы на этом этапе мы проводили измерения в стандартном базисе, эта суперпозиция коллапсировала бы в любое из базисных состояний с той же вероятностью. Таким образом, наши шансы угадать правильное значение находятся в пределах, как и следовало ожидать. Следовательно, в среднем нам нужно будет несколько раз попытаться угадать правильный элемент. Войдите в процедуру, называемую усилением амплитуды, с помощью которой квантовый компьютер значительно увеличивает эту вероятность. Эта процедура расширяет (усиливает) амплитуду отмеченного элемента, что уменьшает амплитуду других элементов, так что измерение конечного состояния вернет правильный элемент с почти достоверностью.

###### Этот алгоритм имеет красивую геометрическую интерпретацию в терминах двух отражений, которые генерируют вращение в двухмерной плоскости. Единственные два особых состояния, которые нам нужно учитывать – это победитель и равномерная суперпозиция. Эти два вектора охватывают двумерную плоскость в векторном пространстве. Они не совсем перпендикулярны, потому что также возникают в суперпозиции с амплитудой. Однако мы можем ввести дополнительное состояние, которое находится в промежутке между этими двумя векторами, которое перпендикулярно и получается путем удаления и изменения масштаба.

###### Шаг 0. Процедура усиления амплитуды начинается с однородной суперпозиции. (Равномерная суперпозиция легко построить из, как было показано в предыдущем разделе.) В начальном состоянии есть.

###### image1

###### Левый рисунок соответствует двумерной плоскости, охватываемой. Правый график представляет собой гистограмму амплитуд состояния для случая. Средняя амплитуда обозначена пунктирной линией.

###### Шаг 1. Применяем к состоянию отражение оракула.

###### image2

###### Геометрически это соответствует отражению состояния около. Это преобразование означает, что амплитуда перед состоянием становится отрицательной, что, в свою очередь, означает, что средняя амплитуда была понижена.

###### Шаг 2. Теперь применим дополнительное отражение о состоянии. В скобках записано это отражение. Это преобразование отображает состояние и завершает преобразование.

###### image3

###### Операцию также можно записать так, чтобы показать, что это проекция на равномерную суперпозицию за вычетом проекции на ее ортогональное дополнение.

###### Два отражения всегда соответствуют вращению. Преобразование поворачивает исходное состояние ближе к победителю. Действие отражения на гистограмме амплитуды можно понимать, как отражение около средней амплитуды. Поскольку средняя амплитуда была понижена первым отражением, это преобразование увеличивает отрицательную амплитуду примерно до трехкратного ее исходного значения, в то время как другие амплитуды уменьшаются. Затем мы переходим к шагу 1, чтобы повторить приложение. Эта процедура будет повторяться несколько раз, чтобы

###### на победителя.

###### После шагов состояние изменится на число применение вращений. Это становится ясно, если посмотреть на амплитуды состояния. Мы видим, что амплитуда растет линейно с количеством заявок. Однако, поскольку мы имеем дело с амплитудами, а не с вероятностями, размерность векторного пространства входит как квадратный корень. Следовательно, в этой процедуре усиливается амплитуда, а не только вероятность.

###### image4

## 3.17.1. Примеры схем

###### Давайте теперь рассмотрим простой пример. Наименьшая схема, для которой это может быть реализовано, включает два кубита, то есть, что означает четыре возможных оракула, по одному на каждый выбор победителя. Первая часть этого примера создает однородную суперпозицию. Вторая часть маркирует состояние с помощью логического элемента control-Z, созданного из CNOT (как описано в последнем разделе). Выполняется заключительная часть схемы.

###### Гровер N = 2 A = 00

###### image5

###### Гровер N = 2 A = 01

###### image6

###### Гровер N = 2 A = 10

###### image7

###### Гровер N = 2 A = 11

###### image8

1. 3.18. Квантовая оценка фазы

###### Квантовая оценка фазы – одна из самых важных подпрограмм в квантовых вычислениях. Он служит центральным строительным блоком для многих квантовых алгоритмов и реализует измерение практически для любого эрмитова оператора. Напомним, что квантовый компьютер изначально позволяет нам измерять только отдельные кубиты. Если мы хотим измерить более сложную наблюдаемую, такую ​​как энергия, описываемая гамильтонианом, мы прибегаем к квантовой оценке фазы. Подпрограмма подготавливает собственное состояние эрмитова оператора в одном регистре и сохраняет соответствующее собственное значение во втором регистре.

###### Схема измерения Джона фон Неймана

###### Квантовая оценка фазы – это дискретизация предписания фон Неймана для измерения эрмитовой наблюдаемой. Схема, которую предвидел фон Нейман, заключается в следующем. Мы рассматриваем квантовую систему, которая поддерживает наблюдаемое, которое мы хотим измерить. Мы предполагаем, что можем измерять только более простые наблюдаемые, в нашем случае отдельные кубиты или, как в исходной настройке, местоположение отдельной частицы.

###### Поэтому наша цель - свести измерение сложной наблюдаемой к измерению более простой наблюдаемой, такой как местоположение. Затем эта простая наблюдаемая называется указателем. Чтобы отобразить сложную наблюдаемую на более простую, мы воспользуемся удобным наблюдением из квантовой механики.

###### Известно, что оператор импульса порождает сдвиги положения одиночных частиц.То есть, если применить унитарность к какому-то волновому пакету, то этот волновой пакет сместится в положительном направлении. Это изображение описывает схему измерения фон Неймана. Теперь мы выполняем шаги и сначала присоединяемся к вспомогательному кубиту – указателю, который представляет собой непрерывную квантовую переменную, инициализированную в состоянии (источнике). Теперь рассмотрим действие этого измерительного прибора.

###### Теперь дискретизированный гамильтониан представляет собой сумму членов, включающих не более кубитов, если это гамильтониан, включающий не более чем кубиты. Таким образом, мы можем моделировать динамику, используя стандартные методы. В терминах собственного базиса импульса записывается начальное (дискретизированное) состояние указателя.

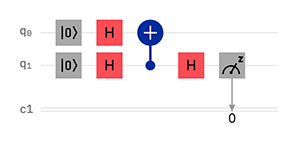
###### Это состояние можно эффективно подготовить на квантовом компьютере, сначала инициализировав кубиты указателя в состоянии и применив (обратное) квантовое преобразование Фурье. Поскольку у нас очень простое начальное состояние, преобразование Фурье может быть представлено произведением матриц Адамара.

###### Выполнение обратного квантового преобразования Фурье указателя оставляет систему в состоянии, в котором наблюдается сильный пик вблизи местоположения. Чтобы гарантировать отсутствие ошибок переполнения, нам нужно выбрать. Легко видеть, что фактическое выполнение моделирования for является продуктом моделирования эволюции в соответствии с единицами времени. Чтобы реализовать полную схему на квантовом компьютере, нам все еще нужно разложить управляемые унитары, а также обратное квантовое преобразование Фурье, обозначенное как, в наши элементарные вентили.

###### Пример ниже демонстрирует квантовую оценку фазы для игрушечного однокубитного гамильтониана, действующего на кубит. Кубит служит системой указателей. Например, квантовое преобразование Фурье в системе указателей эквивалентно вентилю Адамара. Дискретизированная эволюция системы + указатель описывается вентилем CNOT. Окончательный результат измерения кубита–указателя зависит от того, подготовлен он в собственном или в собственном состоянии.

## 3.18.1. Схема оценки фазы (+)

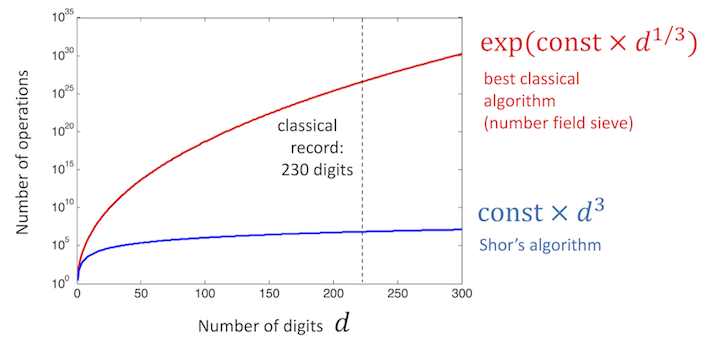
Здесь кубит инициализируется в состоянии, которое является собственным вектором.



1. 3.19. Алгоритм Шора

###### Хотя любое целое число имеет уникальное разложение в произведение простых чисел, поиск простых множителей считается сложной задачей. Фактически, безопасность наших онлайн-транзакций основана на предположении, что факторинг целых чисел с тысячей и более цифрами практически невозможен. Это предположение было оспорено в 1995 году, когда Питер Шор предложил квантовый алгоритм с полиномиальным временем для решения проблемы факторизации. Алгоритм Шора, возможно, является наиболее ярким примером того, как парадигма квантовых вычислений изменила наше представление о том, какие проблемы следует считать решаемыми. В этом разделе мы кратко суммируем некоторые основные факты о факторинге, выделяем основные составляющие алгоритма Шора и проиллюстрируем, как он работает, с помощью игрушечной задачи факторинга.

###### Предположим, наша задача состоит в том, чтобы разложить целое число на десятичные цифры. Алгоритм грубой силы перебирает все простые числа до и проверяет, делятся ли они. В худшем случае это займет примерно время, которое экспоненциально зависит от количества цифр. Более эффективный алгоритм, известный как квадратичное решето, пытается построить целые числа, кратные.



###### Экспоненциальное масштабирование во время выполнения ограничивает применимость классических алгоритмов факторизации числами из нескольких сотен цифр; мировой рекорд (на который потребовалось около 2000 процессорных лет!). Напротив, алгоритм факторизации Шора имеет многочлен во времени выполнения. Описанная ниже версия алгоритма, созданная Алексеем Китаевым, требует примерно кубитов и имеет примерно время выполнения.

1. 3.20. Период нахождения

###### Математикам было известно с 1970-х годов, что факторинг становится простым, если можно решить другую трудную задачу: найти период модульной экспоненциальной функции. Задача поиска периода определяется следующим образом. Для заданных целых чисел и найдите наименьшее положительное целое число, кратное. Число называется периодом по модулю. Напомним, что в модульной арифметике остаток от деления называется значением по модулю и обозначается. Например, . Таким образом, период по модулю - это наименьшее положительное целое число такое, что. Например, предположим и. затем

###### image1

###### То есть имеет период по модулю. Обратите внимание, что вычисление более высоких степеней приведет к периодической последовательности: для любого целого числа. Таким образом, – период модульной экспоненциальной функции. В общем, задача нахождения периода хорошо определена, если и являются взаимно простыми (не имеют общих факторов).

1. 3.21. От факторинга к поиску периода

###### Предположим на мгновение, что нам дана машина для определения периода, которая принимает на вход простые числа и выдает период по модулю. Теперь мы покажем, как использовать машину, чтобы найти все простые множители.

###### Сначала выберите случайное целое число между и и вычислите наибольший общий делитель (НОД). Это можно сделать очень эффективно, используя алгоритм Евклида. Если нам повезет, и у нас есть общие простые множители, и в этом случае gcd равно или, то все готово. С этого момента предположим, что НОД, то есть, и взаимно просты. Позвольте быть периодом по модулю, вычисленным машиной. Повторите вышеуказанные шаги с различными случайными вариантами, пока не станет четным. Можно показать, что значительная часть всех целых чисел имеет четный период (примеры см. В Таблице 1), поэтому в среднем требуется всего несколько повторений.

###### image4

###### Таблица 1: период целых чисел mod

###### На данный момент мы нашли некоторую четную пару, которая является наименьшим целым числом, кратным. Воспользуемся тождеством

###### image3

###### Приведенное выше показывает, что это не кратно (иначе период был бы равен). Предположим на мгновение, что оно не кратно. Тогда ни одно из целых чисел не является кратным, но их произведение кратно. Это возможно только в том случае, если является простым делителем и является простым делителем (или наоборот). Таким образом, мы можем найти и вычислить gcd; см. примеры в таблице 1. В оставшемся «неудачном» случае, когда оно кратно, мы отказываемся от

1. 3.22. Контролируемые операции и оценка фаз

###### Позвольте быть модульной операции умножения. На данный момент мы знаем, как построить квантовую схему, реализующую, а также повторяющиеся квадраты. Мы также знаем, что собственные значения отображают информацию о периоде по модулю. Последний шаг – измерить собственные значения. Для этого нам нужна управляемая версия. Управляемая унитарная операция – это квантовый аналог классических условных операторов, таких как if-then-else. В общем, предположим, что это квантовая схема, действующая на кубиты. Управляемая версия – это унитарная операция, действующая в более крупной системе. Controlled – применяется к целевому регистру, если есть контрольный кубит, и ничего не делает, если контрольный кубит есть.

###### image12

###### Как и их классические аналоги, управляемые квантовые операции используются практически в любом квантовом алгоритме. Заметим, что если может быть реализовано короткой квантовой цепью, то можно и управляемо. Действительно, можно взять реализующую схему и заменить каждый вентиль его управляемой версией (с тем же управляющим кубитом). Основное отличие от классической конструкции if-then-else заключается в том, что управляемый кубит может находиться в суперпозиции состояний. Можно сказать, что в квантовом мире две ветви условного оператора могут выполняться «одновременно». Теперь рассмотрим особый случай, когда целевой регистр подготовлен в некотором состоянии, которое является собственным вектором, т.е. Единственная разница между двумя ветвями управляемого режима – это фазовый сдвиг. Другими словами, контрольный кубит отображается из в, в то время как целевой регистр остается в этом состоянии. Таким образом, мы можем описать действие управляемого элемента на составную систему посредством однокубитового элемента с фазовым сдвигом, действующего на управляющий кубит.

###### image12

Ниже мы сосредоточимся на том, что происходит только с контрольным кубитом (имея в виду, что он является частью более крупной системы). Легко проверить, что вероятность наблюдения результата измерения есть для первой цепи и для второй цепи. Имейте в виду, что это представляет собой управляемую операцию, поэтому схема извлекает информацию о фазе путем измерения интерференции между двумя ветвями управляемого, где одна ветвь накапливает фазовый коэффициент, а другая ветвь не накапливает фазы. Повторяя каждую схему несколько раз и собирая статистику измерений, мы можем оценить вероятности, что дает нам оценку. Для конкретности предположим, что мы готовы выполнить не более 100 измерений. Таким образом, статистическая ошибка в нашей оценке составляет примерно 10%. Чтобы разложить на множитель число с 1000 десятичными знаками, фаза должна быть оценена с очень высокой точностью. Напомним, что мы можем эффективно реализовать для очень больших значений классическим вычислением и использованием идентификатора. Поскольку все операции имеют один и тот же собственный вектор, мы можем выполнять все оценки фазы с одним и тем же целевым регистром (инициализированным в собственном векторе. Управляемая операция вызывает сдвиг фазы на угол на управляющем кубите; таким образом, мы можем оценить с точностью до 10%, выполнив примерно 100 измерений. Это дает оценку с точностью до 5%. Точнее, поскольку фаза живет в единичной цепи, мы получаем пару возможных углов, причем один из них приближается с точностью до 5%, а другой очень далек от (приблизительно). Однако мы уже оценили себя с точностью до 10%. Этого достаточно, чтобы выбрать один из возможных углов и. Применение этого аргумента несколько раз индуктивно показывает, что оценки с постоянной точностью (скажем, 10%) достаточно для грубой оценки. В целом нам потребуются приблизительные измерения, что соответствует управляемым операциям модульного умножения. В общем, весы с некоторыми дополнительными множителями дважды логарифмические.

###### Мы еще не объяснили, как инициализировать целевой регистр в собственном векторе. Например, в качестве базового вектора, кодирующего целое число, можно выбрать начальное состояние.

###### image15

###### **Оценка фазы Т**–**затвора**

###### image19

###### –**смещенная оценка фазы Т-образного вентиля**

###### image20

1. КОНТРОЛЬНЫЕ ВОПРОСЫ
2. Опишите основные возможности IBM Quantum Experience, OpenQASM
3. Что такое Circuit Composer?
4. Как создать квантовую схему с определенным количеством гейтов?
5. Опишите схему последовательности действий между программой, QISKit и локальным моделирующим устройством
6. Что такое пониженная чистота состояния кубита
7. Перечислите основные неунитарные операторы и модификаторы
8. Усиление амплитуды, его этапы

# 4. ОПИСАНИЕ ЛАБОРАТОРНЫХ РАБОТ

1. Лабораторная работа №1. Знакомство со средой моделирования и разработки QASM и IBM Quantum Experience. Составление модели квантовой системы
2. **Цель работы.**

Изучение общих принципов программирования в среде QASM и IBM Quantum Experience. Создание квантовой схемы, которая объединяет кубиты в устройство выполнения логических операций, и применение квантовых вентилей к кубитам для достижения желаемого результата. Предполается создание одного или нескольких кубитов и классических регистров для их измерения в классических регистрах и запуск на моделирующем или реальном квантовом устройстве.

1. **Базовые теоретические сведения.**

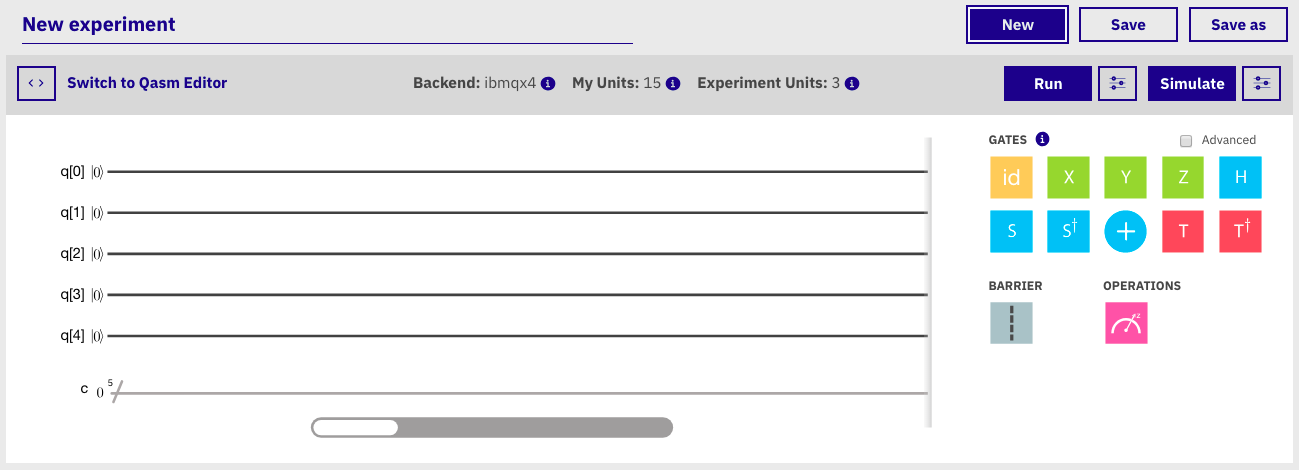
Понимание основ квантового компьютинга, синтаксиса и назначения базовых конструкций языка квантового программирования и облачной среды IBM Quantum Experience.

1. **Задание на выполнение лабораторной работы.**

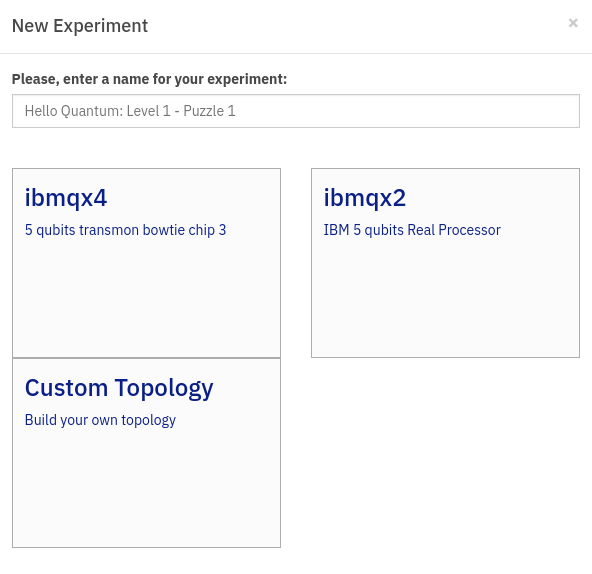
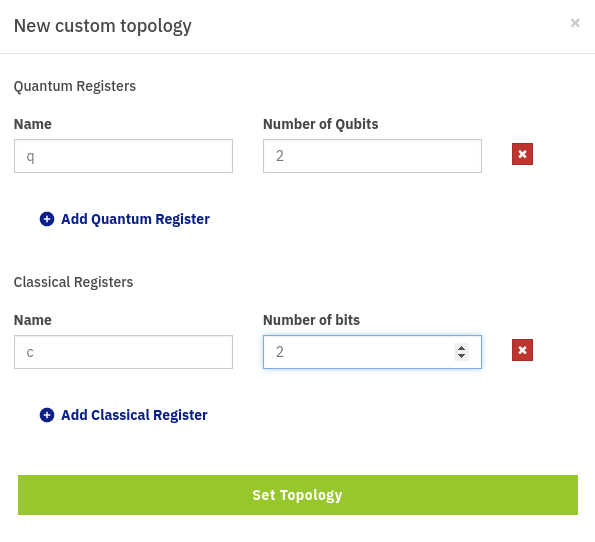
Рассмотреть структуру квантовой программы на примере. В нем необходимо создать один кубит, один классический регистр для его из­мерения, затем применить вентиль Паули X (битовое инвертирование) к кубиту и, наконец, измерить его значение. Основной псевдокод про­граммы можно воспроизвести следующим образом.

1. Создать квантовую программу.
2. Создать один или несколько кубитов и классических регистров для и х измерения.
3. Создать схему, которая объединит кубиты в устройство выполнения логических операций.
4. Применить квантовые вентили к кубитам для достижения желаемого результата.
5. Измерить кубиты в классических регистрах, чтобы получить конечный результат.
6. Скомпилировать программу. На данном шаге создается JSON-представление программы в специфическом формате.
7. Запустить на моделирующем или реальном квантовом устройстве.
8. Получить результаты и проанализировать их.
9. **Пример выполнения**

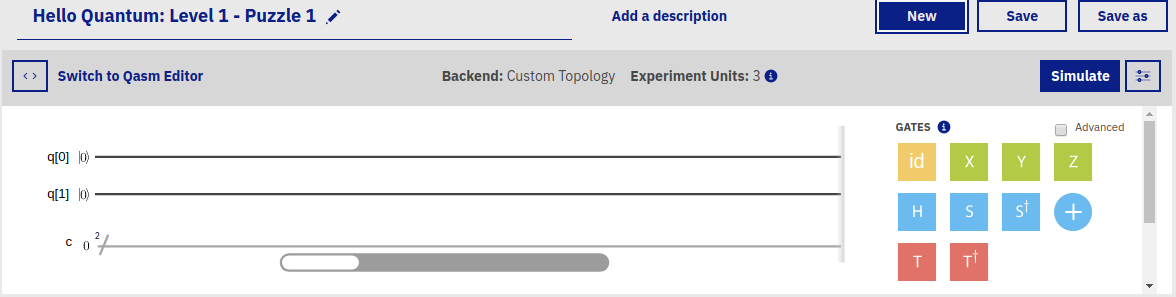
Чтобы перейти к IBM Q Experience, все, что вам нужно сделать, это щелкнуть здесь. Вы попадете в раздел, известный как композитор, который должен выглядеть, как на изображении ниже.



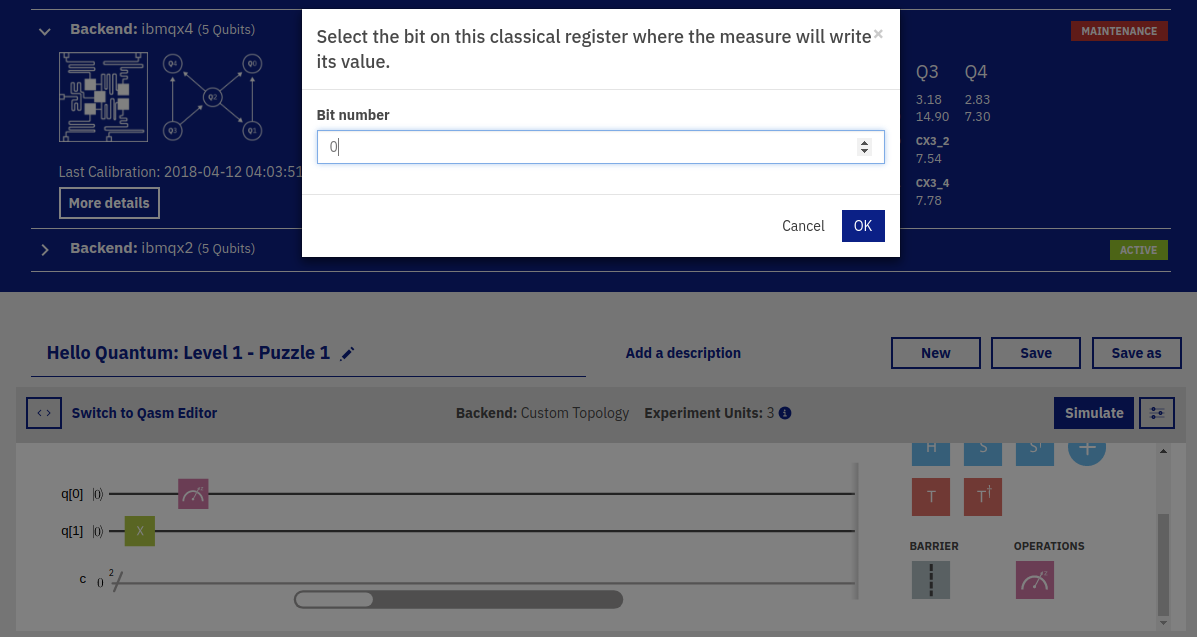
По умолчанию он начинается с пяти кубитов реального квантового процессора, известного как ibmqx4. Настроим композитор всего на два кубита. Для этого мы сначала нажимаем кнопку «Создать» в правом верхнем углу. Затем нам нужно выбрать имя. Нажимаем «Custom Topology» внизу.

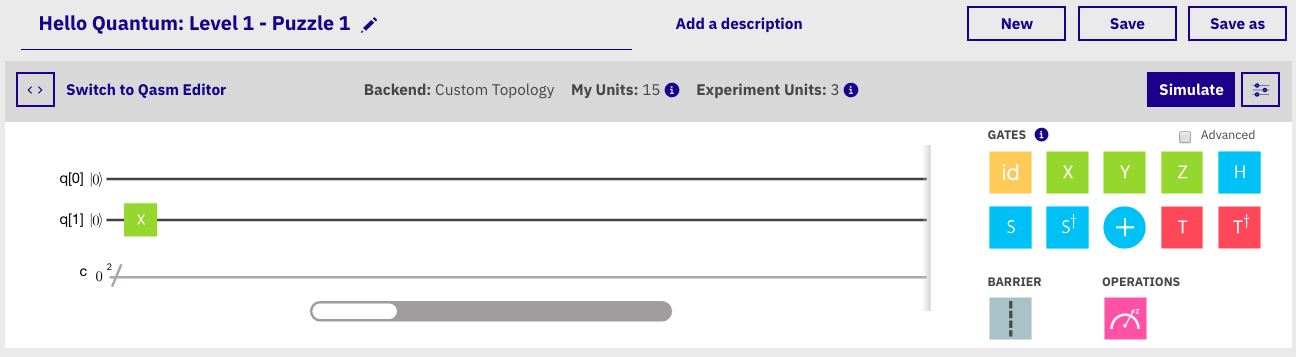
Наконец, нам нужно изменить количество кубитов и битов, которые будут использоваться. В IBM Quantum рассмотрим процессы, в которых два кубита превращаются в два бита, и поэтому нам нужно два кубита. Как только это будет сделано, мы нажимаем «Установить топологию» внизу. Теперь у нас есть композитор, с которым можно поиграть.



В прямоугольниках справа представлены различные операции, которые мы можем выполнять с нашими кубитами. Все, что нам нужно сделать, это перетащить их на линии. Чтобы настроить начальное состояние первой схемы, нам нужно перетащить и X на кубит 1. Теперь посмотрим, сработало ли оно так, как мы ожидали. Первая схема выглядит как на изображении ниже. Нижний кружок для кубита слева белый, что означает, что он с уверенностью выводит 1. Нижний кружок правого кубита черный, что означает, что он определенно равен 0. Мы получаем выходные данные для нижних кружков с помощью измерительного строба. Это можно найти, прокрутив вниз в разделе гейт справа. Чтобы посмотреть на нижний кружок обоих, мы просто перетащим измерительный элемент на обе линии. Когда мы это сделаем, он спросит, какое имя мы дадим получившемуся биту.



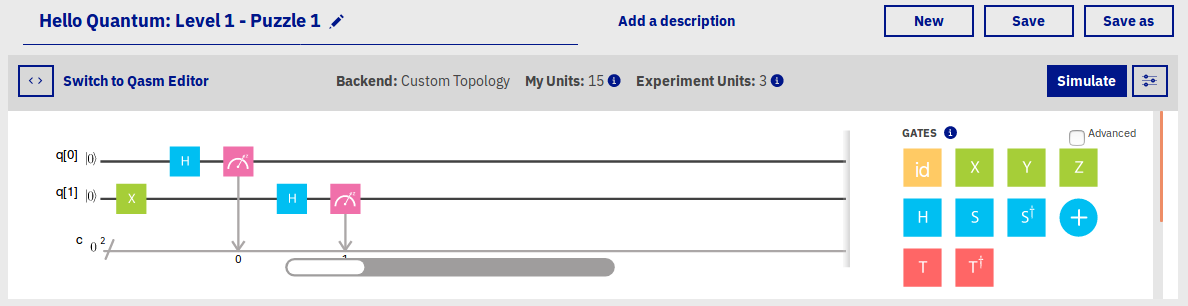
Как только оба гейта будут на месте, можно получить результаты. Сделайте это, нажав «Симулировать».



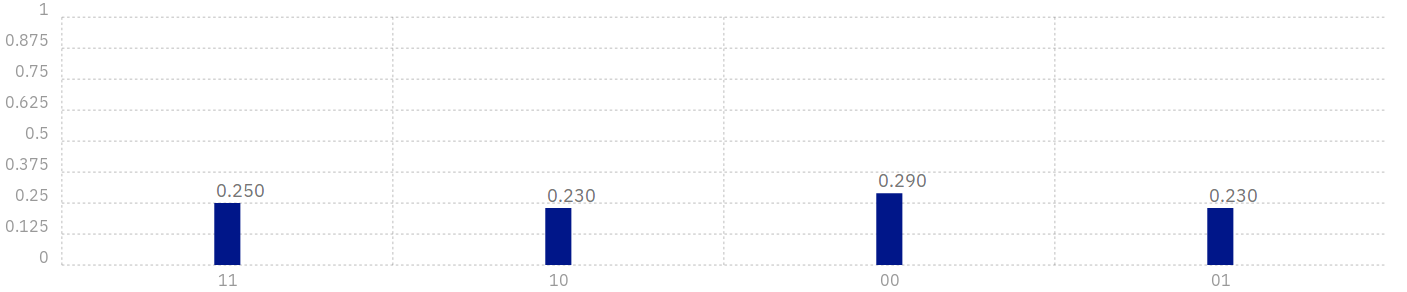
Если вы еще не вошли в систему, вам нужно будет сделать это здесь. Вы можете создать учетную запись только для IBM Q Experience или войти в систему, используя свои существующие учетные записи в социальных сетях. Как только администратор будет готов, вы увидите экран результатов. Это представляет ваши результаты в виде гистограммы.



В этом случае возможен только один результат. Глядя под полоску, мы видим, что это результат 10, что означает результат 1 для левого кубита и 0 для правого. Как и ожидалось! Теперь давайте посмотрим на самые популярные круги. Для этого нам нужно изменить нашу предыдущую схему, которые мы забиваем кубитам.



Теперь у нас есть гистограмма.



Теперь есть четыре возможных результата: 00, 01, 10 и 11. Симулятор запускает процесс 100 раз и обнаруживает, что 11 выходит в 25% случаев, 10 - в 23% и так далее. Если вы запустите это самостоятельно, вы почти наверняка найдете другие числа. На самом деле все они должны получиться с одинаковой вероятностью 25%. Но поскольку мы используем только конечное число выборок, мы всегда будем видеть подобные статистические аномалии. В любом случае мы видим, что оба кубита случайным образом дают ответ 0 или 1.

1. **Содержание отчета**
   1. Титульный лист.
   2. Цель и назначение работы.
   3. Теоретическая часть.
   4. Практическая часть, ход выполнения.
   5. Выводы.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Ограничения, налагаемые на классические компьютеры при решении задач. Идея квантового компьютера
2. Кубит и его динамика. Сфера Блоха
3. Возможные конструкции квантовых компьютеров
4. Программное обеспечение для квантового компьютера
5. Квантово-механические основания информатики (матрица плотности, ее свойства)
6. Информационная энтропия квантового ансамбля (энтропия фон Неймана)
7. Взаимная информация. Информация Холево
8. Основные понятия алгебры логики и логические вентили
9. Логические однокубитовые гейты. Гейт Адамара
10. Контролируемые двухкубитовые квантовые гейты
11. Лабораторная работа №2. Выполнение эффекта квантовой телепортации в среде моделирования и разработки QASM и IBM Quantum Experience
12. **Цель работы.**

Изучение общих принципов программирования в среде QASM и IBM Quantum Experience. Создание квантовой схемы выполнения процедуры квантовой телепортации. Получение теоретических основ и практических навыков создания запутанных передаваемых квантовых состояний и методов шифрования информации.

1. **Базовые теоретические сведения.**

Понимание основ квантового компьютинга, синтаксиса и назначения базовых конструкций языка квантового программирования и облачной среды IBM Quantum Experience.

1. **Задание на выполнение лабораторной работы.**

Рассмотреть структуру построенной квантовой программы для реализации процедуры квантовой телепортации. В нем необходимо добавить в квантовую схему определенное количество кубит, один классический регистр, затем применить гейты к кубитам и произвести процедуру квантовой телепортации.

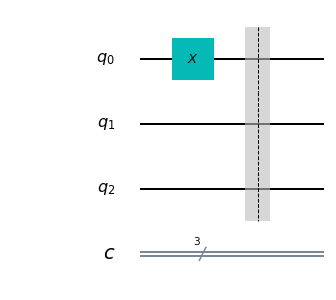
**Пример выполнения**

Квантовая информационная телепортация из кубита 0 в кубит 2

***1. Приготовить кубит (кубит 0, q0)***

После импорта необходимых библиотек начинаем с создания 3 кубитов (3 регистра квантовых битов): q0, q1 и q2 (а также 3 классических регистра для хранения наших измерений кубитов позже). Затем к q0 применяется X-вентиль, чтобы изменить его начальное состояние с |0> на |1>. q0 – исходный кубит, в котором его квантовое состояние (|1>) будет перенесено в целевой кубит q2 через q1.

from qiskit import \*  
%matplotlib inline  
qr = 3 # quantum bit register  
cr = 3 # classical bit register  
circuit = QuantumCircuit(qr, cr)  
circuit.draw(output='mpl’)  
# flip qubit 0 from initial |0> to |1>  
circuit.x(0)  
circuit.barrier()  
circuit.draw(output='mpl', scale=1)



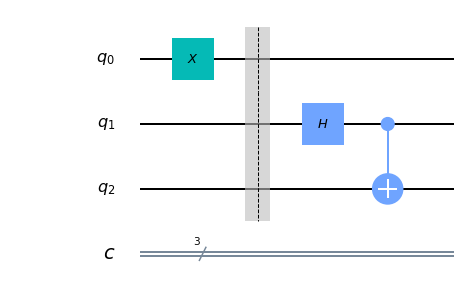
Обратите внимание, что для лучшего чтения установлен барьер. Он также говорит транспилятору не оптимизировать нашу квантовую схему. Транспилятор – это своего рода компилятор для преобразования нашей квантовой схемы в оптимизированную квантовую схему для лучшего выполнения в квантовом компьютере. Транспиляция – это процесс, при котором квантовая схема преобразуется в новую квантовую схему, которая выполняет ту же задачу, но реструктурируется для обеспечения совместимости с физической структурой конкретной квантовой системы и по возможности оптимизировать его работу.

***2. Создайте протокол квантовой телепортации***

Мы запускаем протокол квантовой телепортации, создавая сцепление между q1 и q2.

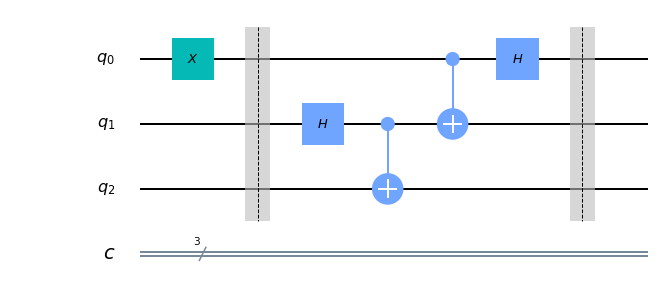
Создайте суперпозицию, применив (H-вентиль) гейт Адамара к q1, добавьте вентиль Controlled-X (CX) между q1 и q2 (q1 как управляемый кубит, q2 как целевой кубит), чтобы создать запутанность.

circuit.h(1)  
circuit.cx(1,2)  
circuit.draw(output='mpl', scale=1)



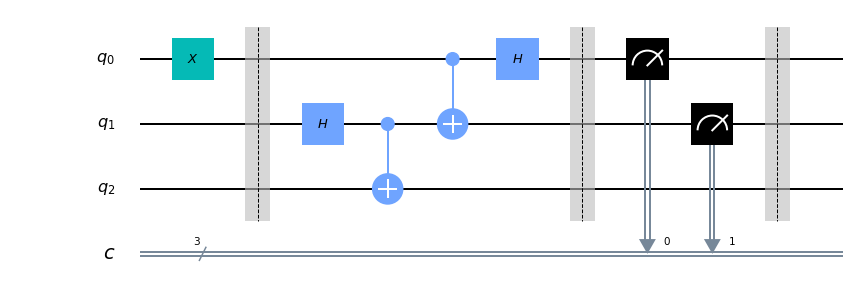
Добавьте вентиль CX между q0 и q1 (q0 как управляемый кубит, q1 как целевой кубит), примените H-образный вентиль на q0, добавьте еще один барьер для ясности.

circuit.cx(0,1)  
circuit.h(0)  
circuit.barrier()  
circuit.draw(output='mpl', scale=1)



После этого мы выполняем квантовые измерения на первых двух кубитах: q0 и q1, а затем сохраняем измерения в классических регистрах c0 и c1 соответственно, как показано ниже.

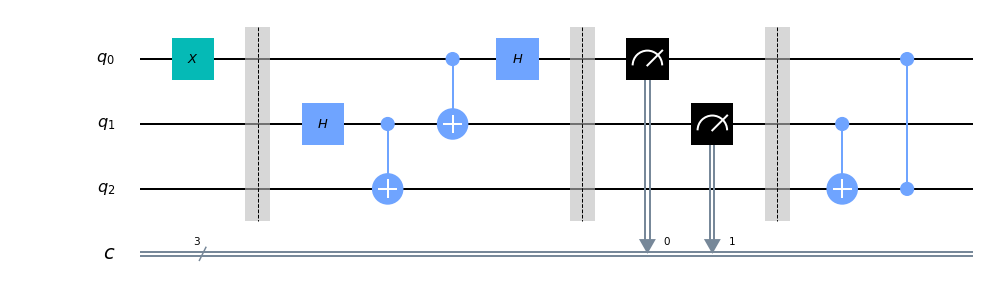
circuit.measure([0,1], [0,1])  
circuit.barrier()  
circuit.draw(output='mpl', scale=1)



***3. Проверьте телепортированную квантовую информацию (путем измерения кубита 2, q2)***

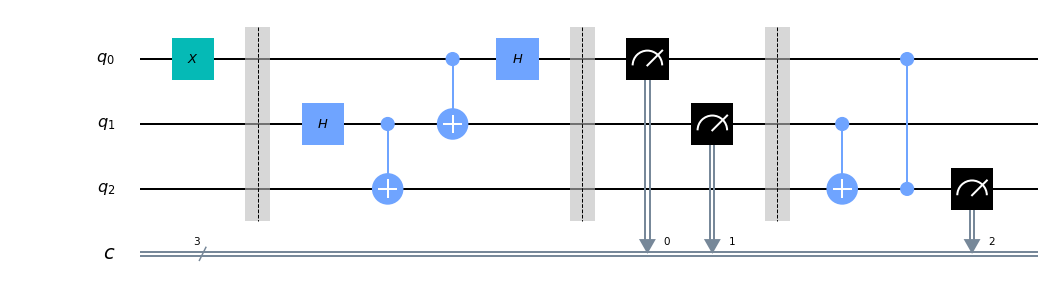
Затем мы применяем CX-вентиль между q1 и q2, как показано ниже (q1 как управляемый кубит, q2 как целевой кубит), а затем CZ-вентиль между q0 и q2 (q0 как управляемый кубит, q2 как целевой кубит).

circuit.cx(1,2)  
circuit.cz(0,2)  
circuit.draw(output='mpl', scale=1)



Наконец, мы выполняем квантовое измерение q2 и сохраняем результат в классическом регистре c2, как показано ниже.

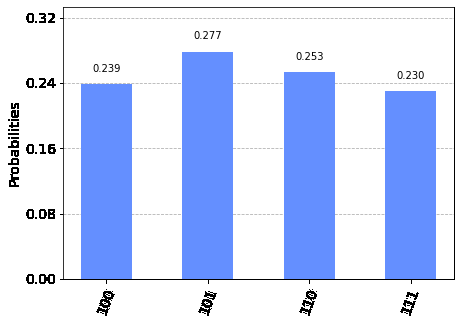
teleported from qubit 0 to qubit 2  
circuit.measure(2,2)  
circuit.draw(output='mpl', scale=1)



Сконструированная квантовая схема затем выполняется на обоих имитаторах квантовых компьютеров (локальном классическом компьютере и реальном квантовом компьютере в IBM Quantum Computing Experience в IBM Cloud). Затем мы проверяем результат выполнения. Результат гистограммы при выполнении нашей схемы на локальном имитаторе квантового компьютера (рассматриваемом как совершенный квантовый компьютер) показывает, что все состояния в q2 находятся в состоянии |1> (как показано ниже) и имеют результаты (количество) измерений (в 3-кубитных последовательностях q2 q1 q0). Мы видим результат для 100, 101, 110 и 111 как 23,9%, 27,7%, 25,3% и 23,0% соответственно). Обратите внимание, что проценты показаны в пропорции 1024, поскольку мы устанавливаем количество снимков: 1024 (фактические значения: “{'110': 259, '100': 245, '101': 284, '111': 236 }”).

from qiskit.tools.visualization import plot\_histogram  
simulator = Aer.get\_backend('qasm\_simulator')  
result = execute(circuit, backend=simulator, shots=1024).result()  
counts = result.get\_counts()  
print(counts)  
plot\_histogram(counts)

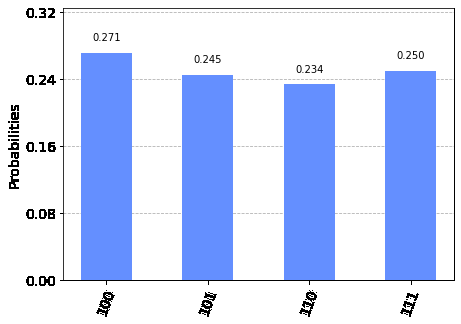
{'110': 259, '100': 245, '101': 284, '111': 236}

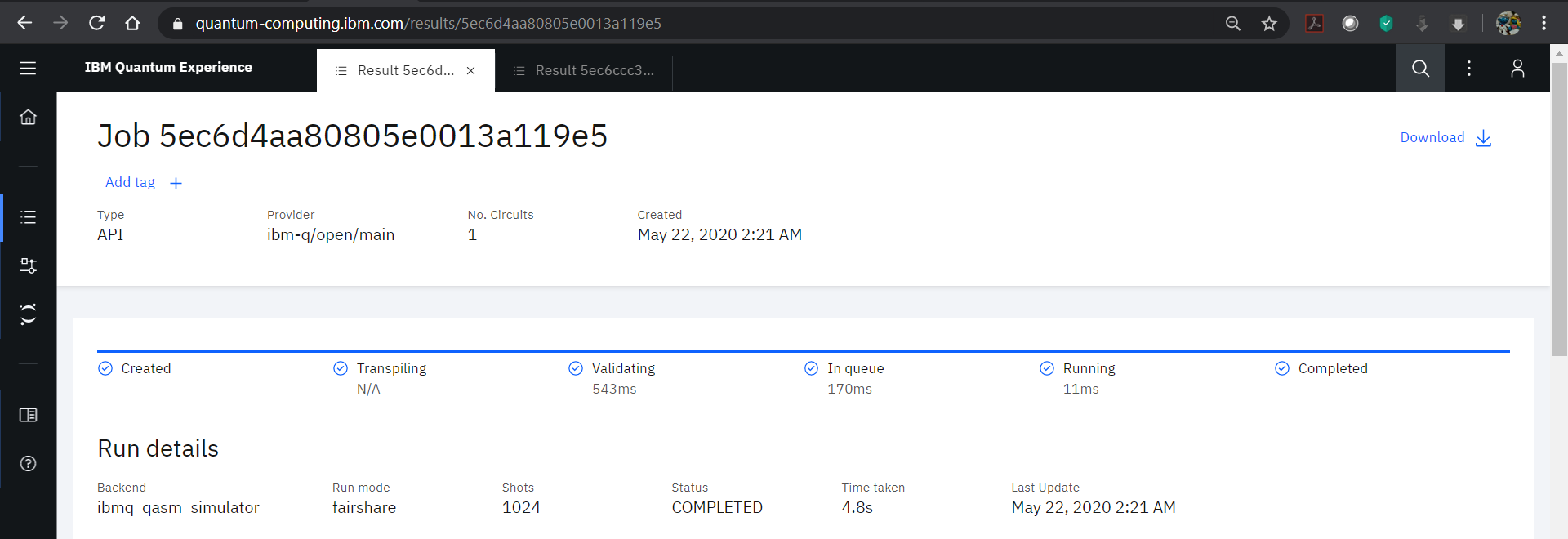


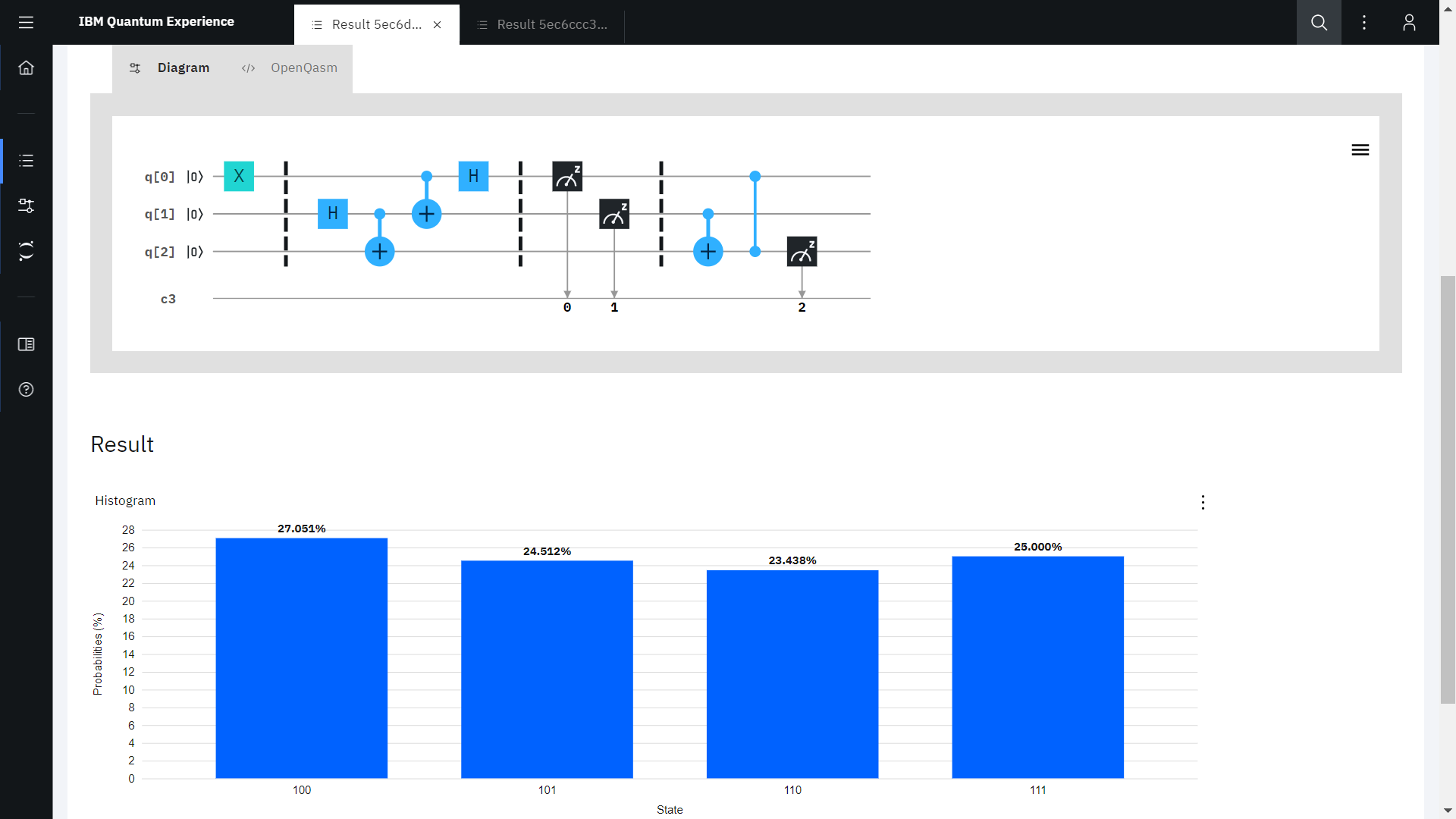
Результат выполнения нашей квантовой схемы в квантовом симуляторе локального компьютера. Результат гистограммы в результате выполнения нашей схемы на квантовом компьютерном симуляторе в IBM Quantum Computing Experience (рассматриваемый как совершенный квантовый компьютер) показывает, что все состояния в q2 находятся в состоянии |1> и имеют результаты (количество) из измерения (в 3 последовательности кубитов q2 q1 q0). Мы видим результат для 100, 101, 110 и 111 с более значимыми 3-значными десятичными знаками как 27,051%, 24,512%, 23,438%, 25,000% соответственно). Фактическое количество остается прежним: “{‘110 ’: 259,‘100’: 245,‘101’: 284,‘111’: 236}”.

from qiskit.tools.monitor import job\_monitor  
IBMQ.load\_account()  
provider = IBMQ.get\_provider('ibm-q')  
qcomp = provider.get\_backend('ibmq\_qasm\_simulator')  
job = execute(circuit, backend = qcomp, shots = 1024)  
job\_monitor(job)  
result = job.result()  
print(counts)  
plot\_histogram(result.get\_counts(circuit))

{'110': 259, '100': 245, '101': 284, '111': 236}







1. **Содержание отчета**
   1. Титульный лист.
   2. Цель и назначение работы.
   3. Теоретическая часть.
   4. Практическая часть, ход выполнения.
   5. Выводы.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Запутывание квантовых состояний. Единица запутанности. Использование запутывания
2. Чистые и смешанные состояния при запутанности
3. Понятие информационной энтропии системы. Количество информации
4. Основные этапы алгоритма квантовой телепортации
5. Методы шифрования информации в алгоритме квантовой телепортации
6. Что такой транспилятор и транспиляция?
7. Квантовый регистр, его место в алгоритме квантовой телепортации
8. Лабораторная работа №3. Реализация алгоритма Дойча-Джозса в среде моделирования и разработки Qiskit и IBM Quantum Experience
9. **Цель работы.**

Изучение общих принципов программирования в среде Qiskit и IBM Quantum Experience. Реализация алгоритма Дойча-Джозса, создание ее квантовой схемы. Получение теоретических основ и практических навыков построения квантовых алгоритмов для решения задач различных классов сложности.

1. **Базовые теоретические сведения.**

Понимание основ квантового компьютинга, синтаксиса и назначения базовых конструкций языка квантового программирования и облачной среды IBM Quantum Experience и Qiskit.

1. **Задание на выполнение лабораторной работы.**

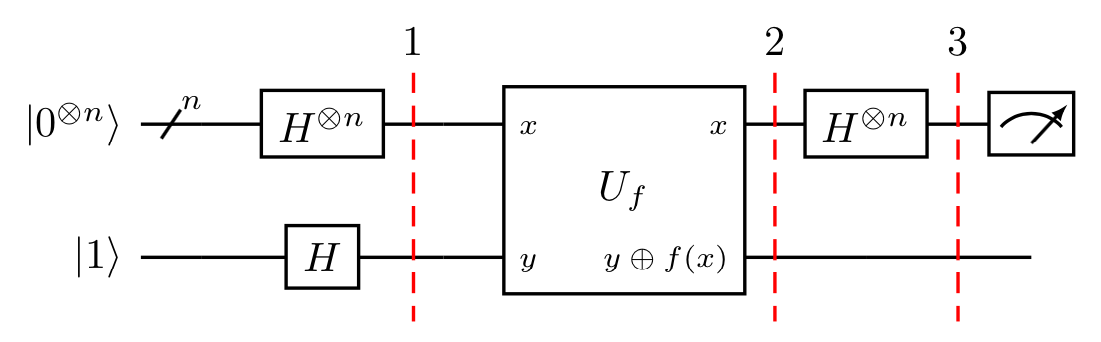
Произвести построение и моделирование работы квантового алгоритма Дойча-Джозса с произвольным набором входных данных. Необходимо добавить в квантовую схему определенное количество кубит, один классический регистр, затем применить гейты к кубитам и произвести процедуру нахождения истинного значения согласно алгоритму Дойча-Джозса. Проанализировать полученные выходные данные и результаты работы алгоритма.

**4. Пример выполнения**

Алгоритм Дойча-Джозса был первым примером квантового алгоритма, который работает лучше, чем лучший классический алгоритм. Он показал, что использование квантового компьютера в качестве вычислительного инструмента для решения конкретной проблемы может иметь преимущества. Нам дана скрытая логическая функция f, которая принимает на входе строку битов и возвращает либо 0 либо 1, то есть:  или 1, где  0 или 1. Свойство данной логической функции состоит в том, что она гарантированно сбалансирована или постоянна. Постоянная функция возвращает все 0 или все 1 для любого входа, а сбалансированная функция возвращает 0 ровно для половины всех входов и 1 для другой половины. Наша задача – определить является ли данная функция сбалансированной или постоянной. Обратите внимание, что проблема Дойча-Джозса n-битовое расширение однобитовой проблемы Дойча. Классически, в лучшем случае, два запроса к оракулу могут определить, есть ли скрытая логическая функция f(x) сбалансированная: например, если мы получим оба f(0,0,0,…) -> 0 и f(1,0,0,…) -> 1, то мы знаем, что функция сбалансирована, поскольку мы получили два разных выхода. В худшем случае, если мы продолжим видеть один и тот же результат для каждого входа, который мы пробуем, нам придется проверить ровно половину всех возможных входов плюс один, чтобы убедиться, что f(x) постоянная. Поскольку общее количество возможных входов равно  это означает, что нам нужно  пробных данных, чтобы убедиться, что f(x) постоянна в худшем случае. Например, для 4-битовой строки, если мы проверили 8 из 16 возможных комбинаций, получив все 0, все еще возможно, что 9-й ввод возвращает 1 и f(x) сбалансированной. Вероятно, это очень маловероятное событие. Фактически, если мы получаем один и тот же результат постоянно подряд, мы можем определить вероятность того, что функция постоянна, как функция от k входов как:



На самом деле, мы могли бы отказаться от нашего классического алгоритма раньше, скажем, если бы мы были уверены более, чем на x%. Но если мы хотим быть уверены на 100%, нам нужно проверить  входы. Используя квантовый компьютер, мы можем решить эту проблему со 100% уверенностью после всего лишь одного вызова функции f(x), если у нас есть функция f реализована как квантовый оракул, отображающий состояние |x⟩|y⟩ к |x⟩|y⊕f(x)⟩, где ⊕ - сложение по модулю 2. Ниже представлена ​​общая схема алгоритма Дойча-Джозса.



Теперь пройдемся по этапам алгоритма:

1. Подготовьте два квантовых регистра. Первый n-кубитовый регистр инициализирован к |0⟩, а второй – однокубитный регистр, инициализированный как |1⟩: 
2. Примените вентиль Адамара к каждому кубиту:



1. Примените квантовый оракул



поскольку для каждого x, f(x) – 0 или 1.

1. На этом этапе второй единичный кубитный регистр можно игнорировать. Примените вентиль Адамара к каждому кубиту в первом регистре:

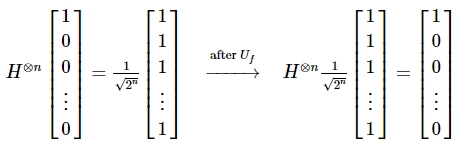
,

где x\*y – сумма побитового произведения.

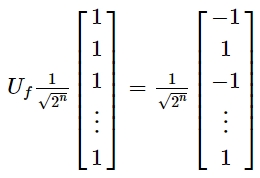
1. Измерьте первый регистр. Обратите внимание, что вероятность измерения

, который оценивается как 1, если f(x) является постоянной функцией и 0, если f(x) является сбалансированной функцией.

Когда оракул постоянен, он не влияет (вплоть до глобальной фазы) на входные кубиты, и квантовые состояния до и после запроса оракула одинаковы. Поскольку H-вентиль является своим собственным обратным, на шаге 4 мы обращаем шаг 2, чтобы получить начальное квантовое состояние |0000…0>.



После шага 2 наш входной регистр представляет собой равную суперпозицию всех состояний в вычислительной базе. Когда оракул уравновешен, фазовый откат добавляет отрицательную фазу ровно к половине этих состояний:



Квантовое состояние после запроса оракула ортогонально квантовому состоянию до запроса оракула. Чт нам, на шаге 4, при применении H-вентилей мы должны получить квантовое состояние, ортогональное |0000…0>. Это означает, что мы никогда не должны измерять состояние «все нули». Давайте рассмотрим конкретный пример двухбитовой сбалансированной функции:

1. Первый регистр двух кубитов инициализируется как |00⟩, а второй регистровый кубит – в |1⟩ (Обратите внимание, что мы используем индексы 1, 2 и 3 для индексации кубитов. Индекс «12» указывает состояние регистра, содержащего кубиты 1 и 2)



1. Примените гейт Адамара ко всем кубитам



Функция оракула может быть реализована как .





1. Упрощая это, мы получаем следующее:



1. Примените гейт Адамара к первому регистру .

Измерение первых двух кубитов даст ненулевой битовый результат – 11, что указывает на сбалансированную функцию. Вы можете попробовать похожие примеры, используя код ниже. Нажимайте кнопки, чтобы добавить H-гейты и оракулы, повторно запустить ячейку и/или установить case = "constant", чтобы попробовать разные оракулы.

from qiskit\_textbook.widgets import dj\_widget

dj\_widget(size**=**"small", case**=**"balanced")

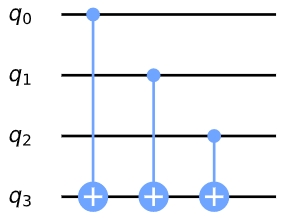
**Создание квантовых оракулов**

Давайте посмотрим, как можно создать квантовый оракул. Для постоянной функции это просто:

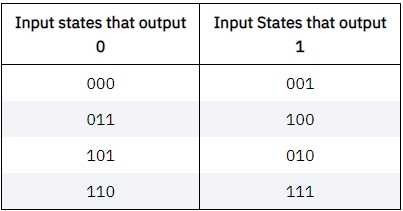
1. Если f (x) = 0, то применить I-гейт к кубиту в регистре 2.

2. Если f (x) = 1, то применить X-гейт к кубиту в регистре 2.

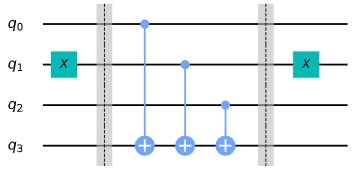
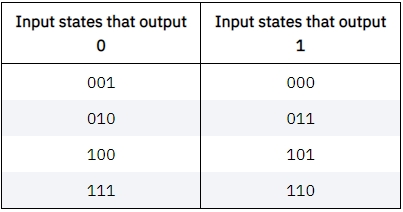
Для сбалансированной функции мы можем создать множество различных схем. Один из способов гарантировать, что наша схема сбалансирована – это выполнение CNOT для каждого кубита в регистре 1 с кубитом в регистре 2 в качестве цели. Например:



На изображении выше три верхних кубита образуют входной регистр, а нижний кубит – выходной регистр. Мы можем увидеть, какие состояния ввода дают какой вывод в таблице ниже:



Мы можем изменить результаты, сохраняя их сбалансированность, заключив выбранные элементы управления в X-гейт. Например, как показано ниже.

**Внедрение Qiskit**

Теперь мы реализуем алгоритм Дойча-Джозса для примера трехбитовой функции с постоянными и сбалансированными оракулами. Сначала сделаем наш импорт:

*# initialization*

import numpy **as** np

*# importing Qiskit*

from qiskit import IBMQ, BasicAer

from qiskit.providers.ibmq import least\_busy

from qiskit import QuantumCircuit, execute

*# import basic plot tools*

from qiskit.visualization import plot\_histogram

Затем мы устанавливаем размер входного регистра для нашего оракула:

*# set the length of the n-bit input string.*

n **=** 3

**Постоянный оракул**

Давайте начнем с создания постоянного оракула, в этом случае ввод не влияет на вывод, поэтому мы просто случайным образом устанавливаем выходной кубит равным 0 или 1:

*# set the length of the n-bit input string.*

n **=** 3

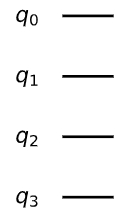
const\_oracle **=** QuantumCircuit(n**+**1)

output **=** np**.**random**.**randint(2)

**if** output **==** 1:

const\_oracle**.**x(n)

const\_oracle**.**draw()



**Сбалансированный оракул**

balanced\_oracle **=** QuantumCircuit(n**+**1)

Далее мы создаем сбалансированный оракул. Можно создать сбалансированный оракул, выполняя CNOT с каждым входным кубитом в качестве элемента управления и выходным битом в качестве цели. Мы можем изменять входные состояния, которые дают 0 или 1, заключая некоторые элементы управления в X-гейта. Давайте сначала выберем двоичную строку длины n, которая определяет, какие элементы управления нужно обернуть:

b\_str **=** "101"

Теперь у нас есть эта строка, мы можем использовать ее как ключ для размещения наших X-гейтов. Для каждого кубита в нашей схеме мы помещаем X-вентиль, если соответствующая цифра в b\_str равна 1, или ничего не делаем, если цифра равна 0.

balanced\_oracle **=** QuantumCircuit(n**+**1)

b\_str **=** "101"

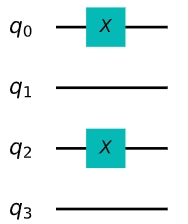
*# Place X-gates*

**for** qubit **in** range(len(b\_str)):

**if** b\_str[qubit] **==** '1':

balanced\_oracle**.**x(qubit)

balanced\_oracle**.**draw()



Затем мы выполняем наши гейты с управляемым НЕ, используя каждый входной кубит в качестве элемента управления, а выходной кубит в качестве цели:

balanced\_oracle **=** QuantumCircuit(n**+**1)

b\_str **=** "101"

*# Place X-gates*

**for** qubit **in** range(len(b\_str)):

**if** b\_str[qubit] **==** '1':

balanced\_oracle**.**x(qubit)

*# Use barrier as divider*

balanced\_oracle**.**barrier()

*# Controlled-NOT gates*

**for** qubit **in** range(n):

balanced\_oracle**.**cx(qubit, n)

balanced\_oracle**.**barrier()

balanced\_oracle**.**draw()

Наконец, мы повторяем код от двух ячеек до завершения обертывания элементов управления в X-вентилях:

balanced\_oracle **=** QuantumCircuit(n**+**1)

b\_str **=** "101"

*# Place X-gates*

**for** qubit **in** range(len(b\_str)):

**if** b\_str[qubit] **==** '1':

balanced\_oracle**.**x(qubit)

*# Use barrier as divider*

balanced\_oracle**.**barrier()

*# Controlled-NOT gates*

**for** qubit **in** range(n):

balanced\_oracle**.**cx(qubit, n)

balanced\_oracle**.**barrier()

*# Place X-gates*

**for** qubit **in** range(len(b\_str)):

**if** b\_str[qubit] **==** '1':

balanced\_oracle**.**x(qubit)

*# Show oracle*

balanced\_oracle**.**draw()



Мы только что создали сбалансированный оракул. Все, что осталось сделать, это посмотреть, может ли алгоритм Дойча-Джозса решить требуемую задачу.

**Полный алгоритм**

Этот первый шаг в алгоритме – инициализировать входные кубиты в состояние |+⟩ и выходной кубит в состоянии |->:

dj\_circuit **=** QuantumCircuit(n**+**1, n)

*# Apply H-gates*

**for** qubit **in** range(n):

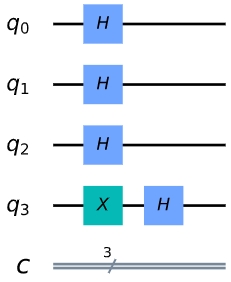
dj\_circuit**.**h(qubit)

*# Put qubit in state |->*

dj\_circuit**.**x(n)

dj\_circuit**.**h(n)

dj\_circuit**.**draw()



Далее применим оракул. Здесь мы применяем Balanced\_oracle, который создали выше:

dj\_circuit **=** QuantumCircuit(n**+**1, n)

*# Apply H-gates*

**for** qubit **in** range(n):

dj\_circuit**.**h(qubit)

*# Put qubit in state |->*

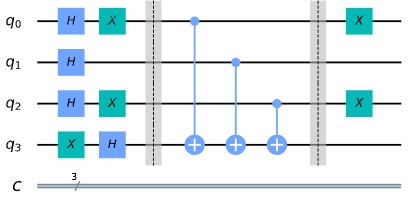
dj\_circuit**.**x(n)

dj\_circuit**.**h(n)

*# Add oracle*

dj\_circuit **+=** balanced\_oracle

dj\_circuit**.**draw()



Наконец, мы выполняем H-вентили на n входных кубитов и измерить наш входной регистр:

dj\_circuit **=** QuantumCircuit(n**+**1, n)

*# Apply H-gates*

**for** qubit **in** range(n):

dj\_circuit**.**h(qubit)

*# Put qubit in state |->*

dj\_circuit**.**x(n)

dj\_circuit**.**h(n)

*# Add oracle*

dj\_circuit **+=** balanced\_oracle

*# Repeat H-gates*

**for** qubit **in** range(n):

dj\_circuit**.**h(qubit)

dj\_circuit**.**barrier()

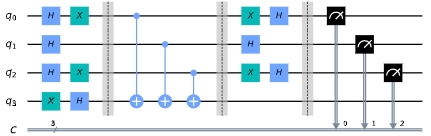
*# Measure*

**for** i **in** range(n):

dj\_circuit**.**measure(i, i)

*# Display circuit*

dj\_circuit**.**draw()



Посмотрим на результат:

*# use local simulator*

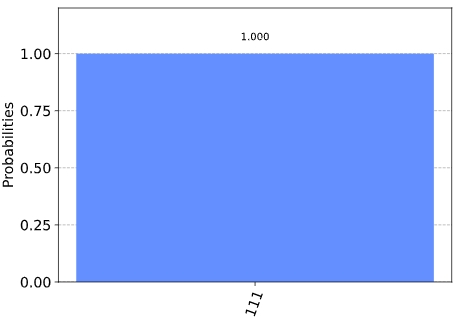
backend **=** BasicAer**.**get\_backend('qasm\_simulator')

shots **=** 1024

results **=** execute(dj\_circuit, backend**=**backend, shots**=**shots)**.**result()

answer **=** results**.**get\_counts()

plot\_histogram(answer)



Из приведенных выше результатов видно, что вероятность измерения 000 составляет 0%. Это правильно предсказывает, что функция сбалансирована.

**Обобщенные схемы**

Ниже представим обобщенную функцию, которая создает оракулы и превращает их в квантовые вентили. Он принимает регистр («сбалансированный» или «константный» и n – размер входного регистра:

**def** **dj\_oracle**(case, n):

*# We need to make a QuantumCircuit object to return*

*# This circuit has n+1 qubits: the size of the input,*

*# plus one output qubit*

oracle\_qc **=** QuantumCircuit(n**+**1)

*# First, let's deal with the case in which oracle is balanced*

**if** case **==** "balanced":

*# First generate a random number that tells us which CNOTs to*

*# wrap in X-gates:*

b **=** np**.**random**.**randint(1,2**\*\***n)

*# Next, format 'b' as a binary string of length 'n', padded with zeros:*

b\_str **=** format(b, '0'**+**str(n)**+**'b')

*# Next, we place the first X-gates. Each digit in our binary string*

*# corresponds to a qubit, if the digit is 0, we do nothing, if it's 1*

*# we apply an X-gate to that qubit:*

**for** qubit **in** range(len(b\_str)):

**if** b\_str[qubit] **==** '1':

oracle\_qc**.**x(qubit)

*# Do the controlled-NOT gates for each qubit, using the output qubit as the target:*

**for** qubit **in** range(n):

oracle\_qc**.**cx(qubit, n)

*# Next, place the final X-gates*

**for** qubit **in** range(len(b\_str)):

**if** b\_str[qubit] **==** '1':

oracle\_qc**.**x(qubit)

*# Case in which oracle is constant*

**if** case **==** "constant":

*# First decide what the fixed output of the oracle will be*

*# (either always 0 or always 1)*

output **=** np**.**random**.**randint(2)

**if** output **==** 1:

oracle\_qc**.**x(n)

oracle\_gate **=** oracle\_qc**.**to\_gate()

oracle\_gate**.**name **=** "Oracle" *# To show when we display the circuit*

**return** oracle\_gate

Давайте также создадим функцию, которая берет эти гейты и выполняет на них алгоритм Дойча-Джозса:

**def** **dj\_algorithm**(oracle, n):

dj\_circuit **=** QuantumCircuit(n**+**1, n)

*# Set up the output qubit:*

dj\_circuit**.**x(n)

dj\_circuit**.**h(n)

*# And set up the input register:*

**for** qubit **in** range(n):

dj\_circuit**.**h(qubit)

*# Let's append the oracle gate to our circuit:*

dj\_circuit**.**append(oracle, range(n**+**1))

*# Finally, perform the H-gates again and measure:*

**for** qubit **in** range(n):

dj\_circuit**.**h(qubit)

**for** i **in** range(n):

dj\_circuit**.**measure(i, i)

**return** dj\_circuit

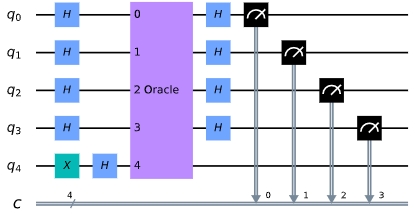
Наконец, давайте воспользуемся этими функциями, чтобы поэкспериментировать с алгоритмом:

n **=** 4

oracle\_gate **=** dj\_oracle('balanced', n)

dj\_circuit **=** dj\_algorithm(oracle\_gate, n)

dj\_circuit**.**draw()

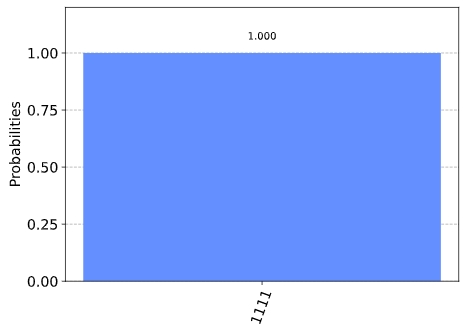


И посмотрите результаты запуска этой схемы:

results **=** execute(dj\_circuit, backend**=**backend, shots**=**1024)**.**result()

answer **=** results**.**get\_counts()

plot\_histogram(answer)



Мы можем запустить схему на реальном устройстве, как показано ниже. Сначала мы ищем наименее загруженное устройство, которое может обрабатывать нашу схему.

*# Load our saved IBMQ accounts and get the least busy backend device with greater than or equal to (n+1) qubits*

IBMQ**.**load\_account()

provider **=** IBMQ**.**get\_provider(hub**=**'ibm-q')

backend **=** least\_busy(provider**.**backends(filters**=lambda** x: x**.**configuration()**.**n\_qubits **>=** (n**+**1) **and** **not** x**.**configuration()**.**simulator **and** x**.**status()**.**operational**==True**))

print("least busy backend: ", backend)

least busy backend: ibmq\_athens

*# Run our circuit on the least busy backend. Monitor the execution of the job in the queue*

from qiskit.tools.monitor import job\_monitor

shots **=** 1024

job **=** execute(dj\_circuit, backend**=**backend, shots**=**shots, optimization\_level**=**3)

job\_monitor(job, interval **=** 2)

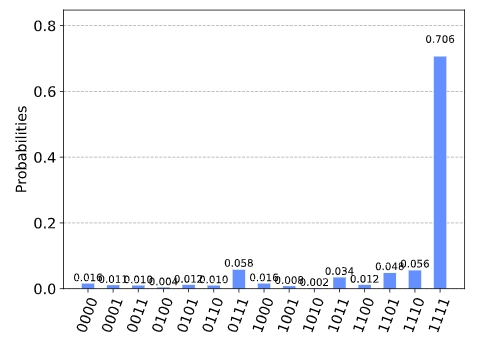
Статус задания: задание успешно выполнено

*# Get the results of the computation*

results **=** job**.**result()

answer **=** results**.**get\_counts()

plot\_histogram(answer)



Как мы видим, наиболее вероятный результат – 1111. Остальные результаты связаны с ошибками в квантовых вычислениях.

1. **Содержание отчета**
   1. Титульный лист.
   2. Цель и назначение работы.
   3. Теоретическая часть.
   4. Практическая часть, ход выполнения.
   5. Выводы.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Типы квантовых ошибок и методы их предотвращения и устранения
2. Алгоритм Дойча-Джозса, особенности реализации и выполнения
3. Сбалансированный и постоянный оракулы, примеры реализации в квантовых схемах
4. Запутывание квантовых состояний. Единица запутанности. Использование запутывания
5. Обратимые и необратимые информационные процессы
6. Булевые функции и их применимость в алгоритме Дойча-Джозса
7. Фазовый запрос, математическая реализация и его место в алгоритме Дойча-Джозса
8. Основные этапы реализации и выполнения алгоритма Дойча-Джозса
9. Лабораторная работа №4. Реализация поискового алгоритма Гровера в среде моделирования и разработки Qiskit и IBM Quantum Experience
10. **Цель работы.**

Изучение общих принципов программирования в среде Qiskit и IBM Quantum Experience. Реализация поискового алгоритма Гровера, создание ее квантовой схемы. Получение теоретических основ и практических навыков построения квантовых алгоритмов для решения задач различных классов сложности.

1. **Базовые теоретические сведения.**

Понимание основ квантового компьютинга, синтаксиса и назначения базовых конструкций языка квантового программирования и облачной среды IBM Quantum Experience и Qiskit.

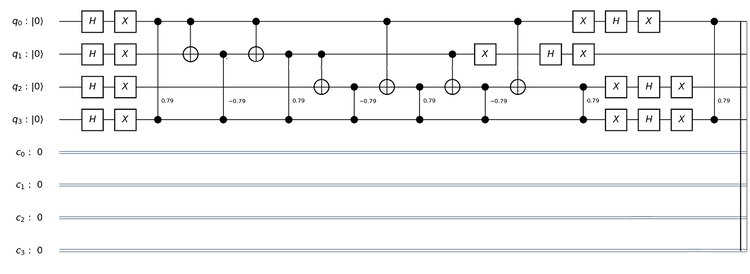
1. **Задание на выполнение лабораторной работы.**

Произвести построение и моделирование работы квантового алгоритма Гровера с произвольным набором входных данных. Необходимо добавить в квантовую схему определенное количество кубит, один классический регистр, затем применить гейты к кубитам и произвести процедуру нахождения истинного значения согласно алгоритму Гровера. Проанализировать полученные выходные данные и результаты работы алгоритма.

**4. Пример выполнения**

Алгоритм Гровера – это алгоритм квантового поиска, который может найти значение или элемент в неотсортированном наборе за O(√N), в отличие от классических алгоритмов поиска, которые в худшем случае найдут элемент за O(N) время. Обратите внимание, что эта реализация представляет собой только одну итерацию. Однако код запускается со 100 снимками, чтобы показать частоту измеренных значений. Алгоритм делится на следующие этапы:

* Поместите все кубиты в суперпозицию, используя вентиль Адамара.
* Реализуйте оракул, который будет отмечать состояние, которое вы хотите найти. Этот оракул работает, выполняя переворот фазы, инвертируя амплитуду состояний.
* Реализуйте схему усиления, которая дополнительно увеличивает амплитуду отмеченных состояний при одновременном уменьшении амплитуды всех других состояний.
* Измерьте все кубиты



***1. Инициализация квантового и классического регистров***

Первый шаг – инициализировать регистр на 4 кубита. Это делается с помощью следующего кода:

q = QuantumRegister(4,'q')

Затем мы инициализируем 4-битный классический регистр следующим кодом:

c = ClassicalRegister(4,'c')

***2. Создайте цепь***

Затем мы создаем квантовую схему, используя следующий код:

circuit = QuantumCircuit(q,c)

***3. Применяйте гейт Адамара для всех кубитов***

Затем нужно применить гейт Адамара. Этот вентиль используется для помещения кубита в суперпозицию 1 и 0, так что при измерении кубита он будет равен 1 или 0 с равной вероятностью. Это делается с помощью следующего кода:

circuit.h(q)

***4. Создайте оракул***

Когда все кубиты находятся в суперпозиции, мы можем реализовать схему оракула. Схема оракула инвертирует амплитуду состояния, делая ее -1/4. Для этого сначала к определенным кубитам применяется вентиль Pauli X (в зависимости от состояния, которое вы хотите отметить). Затем вы применяете Z-гейт с тройным управлением к кубиту 3 со следующим кодом:

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

После этого вы применяете вентиль Pauli X к тем же кубитам, к которым вы применяли вентили раньше. Пример кода оракула для 1100

qc.x(q[0])

qc.x(q[1])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[1])

Чтобы реализовать оракул, который отмечает 1100, сделаем следующее:

* Примените гейт Pauli X к кубитам 0 и 1
* Примените Z-гейт с тройным контролем к кубиту 3
* Снова примените гейт Паули X к кубитам 0 и 1

***5. Создайте схему усиления***

Далее мы применяем схему усиления. Это увеличивает амплитуду помеченного состояния при уменьшении амплитуд всех других состояний. Это делается с помощью следующих шагов:

* Примените гейт Адамара ко всем кубитам
* Примените гейт Pauli X ко всем кубитам
* Примените Z-гейт с тройным контролем к кубиту 3
* Примените гейт Pauli X ко всем кубитам
* Примените гейт Адамара ко всем кубитам

Это делается с помощью следующего кода:

qc.h(q[0])

qc.h(q[1])

qc.h(q[2])

qc.h(q[3])

qc.x(q[0])

qc.x(q[1])

qc.x(q[2])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[1])

qc.x(q[2])

qc.x(q[3])

qc.h(q[0])

qc.h(q[1])

qc.h(q[2])

qc.h(q[3])

***6. Измерьте кубиты***

Последний шаг – измерить все кубиты с помощью следующего кода:

qc.barrier(q)

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

qc.measure(q[1], c[1])

qc.measure(q[2], c[2])

qc.measure(q[3], c[3])

***Как запустить программу***

Скопируйте и вставьте приведенный ниже код в файл Python. Введите свой токен API в части IBMQ.enable\_account ('Вставьте сюда токен API'). Обратите внимание: код содержит все закомментированные оракулы, за исключением оракула, помеченного как 0000. Просто раскомментируйте нужный оракул и запустите.

print('\nGrovers Algorithm')

print('------------------\n')

from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute,IBMQ

import math

from qiskit.tools.monitor import job\_monitor

IBMQ.enable\_account('Enter API token')

provider = IBMQ.get\_provider(hub='ibm-q')

pi = math.pi

q = QuantumRegister(4,'q')

c = ClassicalRegister(4,'c')

qc = QuantumCircuit(q,c)

print('\nInitialising Circuit...\n')

### Initialisation ###

qc.h(q[0])

qc.h(q[1])

qc.h(q[2])

qc.h(q[3])

print('\nPreparing Oracle circuit....\n')

### 0000 Oracle ###

qc.x(q[0])

qc.x(q[1])

qc.x(q[2])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[1])

qc.x(q[2])

qc.x(q[3])

'''

### 0001 Oracle ###

qc.x(q[1])

qc.x(q[2])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[1])

qc.x(q[2])

qc.x(q[3])

'''

'''

### 0010 Oracle ###

qc.x(q[0])

qc.x(q[2])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[2])

qc.x(q[3])

'''

'''

### 0011 Oracle ###

qc.x(q[2])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[2])

qc.x(q[3])

'''

'''

### 0100 Oracle ###

qc.x(q[0])

qc.x(q[1])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[1])

qc.x(q[3])

'''

'''

### 0101 Oracle ###

qc.x(q[1])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[1])

qc.x(q[3])

'''

'''

### 0110 Oracle ###

qc.x(q[0])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[3])

'''

'''

### 0111 Oracle ###

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[3])

'''

'''

### 1000 Oracle ###

qc.x(q[0])

qc.x(q[1])

qc.x(q[2])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[1])

qc.x(q[2])

'''

'''

### 1001 Oracle ###

qc.x(q[1])

qc.x(q[2])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[1])

qc.x(q[2])

'''

'''

### 1010 Oracle ###

qc.x(q[0])

qc.x(q[2])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[2])

'''

'''

### 1011 Oracle ###

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[3])

'''

'''

### 1100 Oracle ###

qc.x(q[0])

qc.x(q[1])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[1])

'''

'''

### 1101 Oracle ###

qc.x(q[1])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[1])

'''

'''

### 1110 Oracle ###

qc.x(q[0])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

'''

'''

###1111 Oracle###

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

'''

print('\nPreparing Amplification circuit....\n')

#### Amplification ####

qc.h(q[0])

qc.h(q[1])

qc.h(q[2])

qc.h(q[3])

qc.x(q[0])

qc.x(q[1])

qc.x(q[2])

qc.x(q[3])

qc.cu1(pi/4, q[0], q[3])

qc.cx(q[0], q[1])

qc.cu1(-pi/4, q[1], q[3])

qc.cx(q[0], q[1])

qc.cu1(pi/4, q[1], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.cx(q[1], q[2])

qc.cu1(-pi/4, q[2], q[3])

qc.cx(q[0], q[2])

qc.cu1(pi/4, q[2], q[3])

qc.x(q[0])

qc.x(q[1])

qc.x(q[2])

qc.x(q[3])

qc.h(q[0])

qc.h(q[1])

qc.h(q[2])

qc.h(q[3])

### Measurment ###

qc.barrier(q)

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

qc.measure(q[1], c[1])

qc.measure(q[2], c[2])

qc.measure(q[3], c[3])

backend = provider.get\_backend('ibmq\_qasm\_simulator')

print('\nExecuting job....\n')

job = execute(qc, backend, shots=100)

job\_monitor(job)

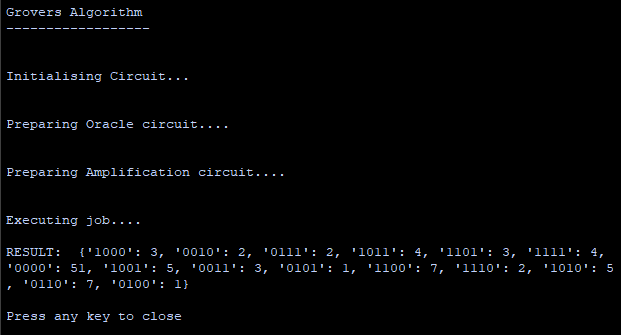
counts = job.result().get\_counts()

print('RESULT: ',counts,'\n')

print('Press any key to close')

input()

После запуска программы вы получите следующий результат:



Вывод отмеченного значения для оракула равного 0000. Обратите внимание на то, что его количество значительно выше, чем у других значений.

1. **Содержание отчета**
   1. Титульный лист.
   2. Цель и назначение работы.
   3. Теоретическая часть.
   4. Практическая часть, ход выполнения.
   5. Выводы.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение понятию квантовому оракулу. Каковы его основные особенности и структура построения в рамках алгоритма Гровера
2. Амплитуда усиления квантовых состояний. Ее назначение и реализация
3. Переворот фазы, как элемент квантовой схемы. Ее назначение в выполнении алгоритма Гровера
4. Обратимые логические гейты
5. Алгоритм Гровера, особенности реализации и выполнения