Kilo-Qubit Scale Quantum Computer Design

Design Document

Team #24
Project for: Dr. Durga Paudyal, Ames Lab
Advisors: Dr. Durga Paudyal, Dr. Jonathan Smith
Team Members:
Nicholas (Nick) Greenwood, Jacob Frieden,
Samuel (Sam) Degnan, Arvid Gustafson,
Chad (Colin) Gorgen, Emile Albert Kum Chi
sdmay23-24@iastate.edu
https://sdmay23-24.sd.ece.iastate.edu

Revised: April 29, 2023

Academic and Professional Utilization

Development Standards & Practices Used

- **Waterfall Design Methodology (Development Practice)**
  - Waterfall Methodology is used because we do not have the knowledge to create many different sprints of designs and will need to put lots of knowledge into a final design.
- **P1730 - Standard for Quantum Computing Definitions (Development Standard)**
  - Through the development of this computer, we must communicate effectively. Adhering to standard definitions will be a must.

Applicable Courses from Iowa State University Curriculum

- "**PHYS 422X/522X: Foundations of Quantum Computing**"
  - Highly relevant class, none of us have taken it
- "**EE432/532: Micro-electronic fabrication Technique**"
- **Other PHYS courses**
  - General physics classes can contain useful information on basic, non-quantum physics

Sources For New Skills/Knowledge Acquired (that was not taught in Iowa State University Curriculum)

- **Quantum Computation and Quantum Information**
  - Textbook by Isaac Chuang and Michael Nielsen
- **Microfabricated Ion Trap Junctions: 3D cross interchange***
  - Paper by Gavin Nop (TA)
- **On Stabilizer Techniques and Their Application to Simulation and Certification of Quantum Devices**
  - Paper on Error Correction
- **Honeywell Ion-Trap Quantum Computer Design Documentation/Review**
  - Presentation by Gavin Nop
- **Computational Physics 4860**
  - University of Northern Iowa physics class
- **Modern Physics 4100**
  - University of Northern Iowa physics class
- **Modern Physics Lab 4110**
  - University of Northern Iowa physics class
- **Various papers, lectures, virtual classes, and Youtube videos on ion traps, quantum computation, error correction, noise and other general quantum terminology and all associated topics discussed in this paper**

Table of Contents

List of figures/tables/symbols/definitions

- **Superposition**
  - Being a combination of both computational basis states (|0> and |1>)
- **Entanglement**
  - One qubit acts in a similar manner to another
- **Quantum Fidelity**
  - Commonly used in quiskit; Represented by capital 'F'
  - (bad) 0 <= F <= 1 (good)
    - An example of 1.0 Fidelity would be finding that collapsing a positive state |+> would result in 50% collapse into 0 and 50% into 1, a perfect distribution
    - If this collapse resulted in always 0s, that a F = 0.5
    - Scales exponentially with # of qubits (2 qubits = 2^2 states)
  - Basically a measure of how closely you can expect your qubit to behave irl compared to in quantum coding
  - Essentially the link between "quantum coding" and actually quantum computing
  - Fidelity < 1 is a result of noise
  - <ψ|A|ψ> = Application of A to ψ, the basis which fidelity is determined off of
- **Quantum Laws**
  - Superposition (see above)
- **Clusters (Quantum and classical meanings)**
  - Set of computers that work together, as such can be viewed as a single system
  - Each node performs the same task
  - Deadlock
    - Two applications are fighting for the same resource, they prevent the other from accessing it, and so both programs cease
    - In single computers (ie nodes), deadlock is handled through algorithms and the OS, which means it probably isn't a big deal
- **Nodes (Quantum and classical meanings)**
  - Each perform the same task in traditional (meaning should be easiest to implement)
  - Each run their own OS
- **Quantum Supremacy**

    - Exactly how it sounds: The goal / idea that a quantum computer can solve a (potentially useless) problem that a classical computer can't in any feasible amount of time
    - Also known as "quantum advantage"
    - Is expected to be done with "near-term" QCs, as the goal doesn't require high-quality error correction or the problem to be useful; no impact or hurdle of commercially viable QCs; primarily scientific
    - Examples of potential problems:
        - Shor's Algo for factoring integers - prime factorization of an n-bit integer in O(n^3) time
        - Boson sampling - usage of boson scattering to evaluate expectation values of permanents of matrices
        - Sampling an output distribution of a quantum circuit - scales difficult exponentially with number of qubits
- **Quantum Volume ("Area")**
    - Metric of capability for a quantum computer, introduced by IBM in 2019
    - Maximum size of square quantum circuits that can be successfully implemented, is always a power of 2
    - As of September 2022, Honeywell's H1 is king at QV 8192
        - 13 square circuits
        - Did this by implementing arbitrary angle two-qubit gates
    - Honeywell has been increasing at a rate of 10x per year
- **Arbitrary angle two-qubit gates**
    - Currently, researchers are working with single qubit gates of fully entangled two qubit gates. Arbitrary angle gates mean we can operate with two partially entangled gates
    - Implemented by Honeywell in their Sept 2022 QV test
    - Instead of having to fully entangle then walk back, can now just add a slight but of entanglement
    - Very useful for fourier transform, where they can have ½ the number of arbitrary angle gates than traditional two qubit gates

Tables:

Figures:

1 Team

The first section provides an overview of the team members, their majors, and the roles that each member had.

1.1 Team Members

- Nicholas Greenwood - Computer Engineering
- Jacob Frieden - Software Engineering
- Sam Degnan - Software Engineering
- Arvid Gustafson - Software Engineering
- Colin Gorgen - Electrical Engineering and Physics
- Emile Albert Kum Chi - Electrical Engineering

1.2 Required Skill Sets for Your Project

A basic understanding of quantum mechanics and a more thorough understanding of quantum computing are required. Knowledge in chip manufacturing techniques, coding techniques for simulation purposes, and robust capability of synthesizing technical writing were all paramount to the success of this project.

1.3 Project Management Style Adopted by the team

We opted for a distributed management style. Nicholas is the contact point for communication between the team and the client / professors and the management of deadlines for documents and other pieces of senior design work. When work items are near due, Nicholas divides up the work evenly among team members.

For all non-senior design work (ie project based work), each team member undertakes their own assignment and sees that is completed to the clients standard. While assignments may overlap, this usually does not become apparent until the team meets again.

1.4 Project Management Roles

- Nicholas Greenwood - Administration and Integration, Hardware Team
- Jacob Frieden - Administration and Research, Software Team
- Sam Degnan - Requirements Synthesis and QC Error Correction expert, Software Team
- Arvid Gustafson - Lead Developer and Coherence expert, Software Team
- Colin Gorgen - Global Hardware Modeler, Hardware Team
- Emile Albert Kum Chi - Electrical Design and QC electronics expert, Hardware Team

2 Introduction

In the introduction we address our problem statement, look at our intended uses and users in depth, and describe what sorts of requirements and constraints accompany our undertaking.

2.1 Problem Statement

We will further the collective knowledge base of quantum computing and computer design by collectively contributing to the design and construction of a working quantum computer at ISU/Ames Lab over the course of two semesters of senior design This project will not conclude with us and will be carried on by future staff and students. We are doing so because quantum computing is a cutting edge technology, which offers opportunities to provide numerous advances in computational and scientific fields, and as a national lab and associated research university, Ames Lab and ISU's goals for furthering the state of science align with their construction of a quantum computer.

2.2 Intended Users and Uses

Ames Lab

- **Characteristics:** homogenous employment; heterogeneous expertise & explicit goals connected to their background, existing proposals/fundings, etc.; Highly technical individuals, likely interested in details and implementation as much as final product;
- **Needs:** Development of new techniques in the design of a quantum computer/proof of concept for existing techniques, verbose documentation/explanation of work
- **How They Use / Benefit:** They will be able to conduct research and forward the current knowledge base on quantum computer design and computing, increasing the productive potential and prestige of the institution.

Iowa State University students and faculty

- **Characteristics:** Large, diverse, and scholastic. A subset of students, likely in ECPE, Physics, ComSci, and related fields will be the most likely to be interested in this product. Within this subset, there are still wildly different areas of knowledge that will correspondingly result in different interests and concerns regarding our product. That said, they will all be technically inclined, though possibly to a lesser degree than the members of Ames Lab, and their access may be comparably limited.
- **Needs:** Access to quantum computing and/or quantum computer design starting at a possibly lower level of technical background than can be assumed of our other user base. This suggests the need for a full bodied "zero to hero" documentation structure.
- **How They Use / Benefit:** Involvement in the quantum computing domain - increases the prestige of the university and the real value offered to its associates through access/exposure to the computer and its design. **Students will have an expanded range of real world projects they can work on and take advantage of.** Will utilize knowledge and any components for furthering of our goal or for new discoveries

State-of-the art researchers:

- **Characteristics:** Continuous drive for improvement, working on science projects. Very open and sharing for the benefit of everyone. Diverse ethnicities, cultural backgrounds, first-languages

- **Needs:** Perform high level calculations, Develop new solutions to current problems using new techniques and technologies developed by themselves or others
- **How They Use / Benefit:** Personal or group glory by using these concepts for further development in the field, enhanced knowledge in the field; Will utilize knowledge and any components for furthering of our goal or for new discoveries

2.3 Requirements & Constraints

Kilo-qubit (scale) Ytterbium Ion-Trap Quantum Computer (QC) Design

Fundamental:

- Design a quantum computer that can be scaled to hold thousand(s) of qubits
- The design should utilize memory ion traps that preserve qubits for longer times (10s of machine cycles). These need to have transport access and be optical hardware addressable
- The design should also utilize computational ion-traps, which are the standard within current ion-trap QC designs.

Resource:

- Mike and Ike quantum physics book
- Honeywell Ion-Trap Quantum Computer Design Documentation/Review
- Papers, lectures, and virtual classes on ion traps and quantum computation
- 3D Modeling software (SolidWorks)
- Qiskit, a development kit for working with quantum circuits on quantum devices or simulations
- Quantum Computer design software
- A suitable word processor (Microsoft Office / G Suite) for documentation

Physical:

- The design of the QC must be of a reasonable size - due temperature constraints associated with operating a quantum computer, all computational hardware must be able to fit within a space that could be cooled to 10 - 12 Kelvin.
- The QC design must be in line with the fabrication capabilities of Sandia[sic] labs, our design implementation collaborators
- QC must be capable of performing low-noise / interference ion transport along the trap.
    - Note: Software based “transport” (swapping) mechanisms exist, but are impractically error-prone.
    - Physical ion-transport is the standard, and minimal ion transport distance is prioritized to decrease error from noise exposure along the transport channel.

Economic / Market:

- There are no economic requirements
- There may be economic constraints
    - The ability to produce QC-level components is not one that Iowa State possesses
    - We would need to utilize outside labor and outside funding to physically build any components
        - Labor in the form of Sandia[sic] labs, our design partner
    - Any future construction of the outlined design would require significant funding in the form of grants or private backing

Software:

- Software requirements
    - We must be able to run quantum circuits accurately
    - We must be able to schedule qubits

    - Quantum resources must be used appropriately, for example you cannot run multiple gates on a qubit in one cycle
    - We must be able to scale the size of the digital twin
- Software constraints
    - Use of ion computer backgrounds in software costs money
    - We do not actually have a quantum computer and can only simulate it
    - We do not have enough time to implement many of the software requirements

Other:

- As is customary in any field of advanced science, an advancement such as ours must be outlined in an academically reviewed paper.

3 Project Plan

Our Project Plan outlines how we completed and tracked our work, decomposes our tasks into actionable steps, and laid out benchmarks / milestones that we achieved and ones that we see as viable going forward.

3.1 Project Management/Tracking Procedures

Due to the large amount of knowledge that we needed to build up, we chose to use a waterfall project management approach. Creating a viable product iteration simply isn't possible within a small portion of time such as a sprint. The requirements of a quantum computer have been well established since the project began. Due to this being a largely physical design (in concept) of this Quantum Computer (QC), the iteration of designs cam in continued implementation of features into an overall computer design, as opposed to doing full minimum viable product (MVPs) and iterating on each one.

We used GroupMe for communication and sharing knowledge amongst student team members. We utilized a mass-email chain for communication amongst all individuals involved in the project. We also have a shared drive with an extensive directory containing our accumulated knowledge base, presentations, design documents, and eventually our designs. The Software team utilized our assigned Git Repository for the creation of their Digital Twin (discussed later). In addition, the Software team organized and attended weekly meetings to discuss softwa-respecific design problems. Finally, we had weekly 2-3 hour meetings with all associated members of the project to talk about the progress that we have made.

3.2 Task Decomposition

- **Knowledge Acquisition: Reading papers, projects, journal, courses, etc. on quantum mechanics and computing. Can be broken into semi-distinct areas of research:**
    - General Quantum Mechanics Knowledge - Mike and Ike book on Quantum
    - Quantum Computer Design - Client Provided State of the Art Review, Honeywell and IonQ whitepapers, etc.
    - Quantum Simulation and Design - IBM Quantum, QISKIT, Quirk, more forthcoming
- **Initial computer design: Decomposing this step was contingent on adequate understanding of quantum computing components. Substeps were:**
    - Defining Ion-trap Design
        - Memory vs. Computation Traps

        - Geometry Selection (2D linear, junctioned with trap inversion)
    - Classical computer control systems
        - Laser system controls (Classical implementation)
        - Electrode controls (Classical implementation)
    - Defining ancillary hardware:
        - Number and makeup of ancillary components
            - Base specific piece of hardware on (limited) qualifying characteristics
        - Positions of components relative to chip and traps
        - Laser Systems:
            - Cooling: Doppler, Simulated Raman, Sideband
            - Trap and ion Initialization
            - Global Addressal
- **Initial software digital twin simulation.**
    - Create architecture for program with the following ideas in mind
        - Optimize scheduling to reduce noise and decoherence
        - Make the architecture easily expanded upon
        - Make architecture uphold to industry standard with functions that do one thing, are named well, and the architecture should be specifically made for the project
    - Implement the following in code:
        - Moving qubits for operations
        - Scheduling qubits appropriately
        - Node level functionality
        - Noise reduction and error correction within these functions
        - Quasi-crystal memory qubits
        - Correctly take input gates and give corresponding output
    - Create unit tests for software
        - Unit tests will test each component. The test shall be specific and uphold to industry standards
        - The tests will be run and shall be adhered to for future development, therefore a focus should be on quality
- **Secondary computer design steps, in tandem with or contingent upon previous steps:**
    - Client functional review: Iteration of specific elements of hardware and software design
    - Solidworks design of known hardware components (Chip and traps) for purposes of spatial reasoning
- **Deliverable creation**
    - Creation of figures and table to aid understanding
    - Lightning Talk presentation creation
    - Mid-Semester and End-Of-Semester presentations
    - Demo video of Digital Twin
    - Drafting of End-of-Semester Reports

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Due to the highly theoretical nature of our project -the construction of a quantum computer- , quantitative or physical milestones are difficult to create:

- **Knowledge Acquisition**
  - This step was purely measured by relative knowledge
  - Knowledge acquisition continued to be the primary step of this entire project, and extended from the first week of 491 through to the last week of 492
  - A good yardstick to measure our success in this step was by our understanding of increasingly complex bodies of knowledge (increasingly complex papers)
  - Around week 10 of CPRE 491, we reached a relative milestone, where our work becomes not solely knowledge acquisition based, and was simply performed as needed
- **Initial Computer Design**
  - Nailing down specific elements of the quantum compute hardware design were important milestones
    - Ion trap size
    - Ion trap orientation on chip
    - Number of and purposes of lasers used for addressal and computation
    - Number of and purposes of ancillary hardware used for addressal and computation
    - Location of laser producers / ancillary hardware relative to chips
  - We did not end up having any metrics / evaluation criteria for the computer design (initial no secondary / further iterations) other than
    - Does it fit in a reasonable packages
    - Can all hardware components and lasers see elements of the trap / chip
    - Within reason, is there any overlap in non-physical spectrums (primary concern was EM)
- **Software**
  - Large software milestones, our software will have:
    - The ability to optimize qubit operation sequences
    - A full suite of unit tests
    - The ability to simulate decoherence and noise
    - Quasi-crystal memory qubits
  - We will evaluate our software based on unit tests, with the following criteria in mind:
    - Our output should have above a 99% fidelity
    - Qubits must have enough coherence time to be used appropriately
    - Qubits must be able to communicate necessary information with other qubits
    - We must be able to scale our overall number of qubits to a kiloqubit level with our computer staying fully functional
  - Non-objective measures:
    - Our code shall be well abstracted
    - Our code shall be well documented
    - Our code shall be easily understandable

- **Secondary computer design steps, in tandem with or contingent upon previous steps:**
    - SolidWorks design of known hardware components (Chip and traps) for purposes of spatial reasoning
        - There were no specific goals of the SolidWorks design other than confirming metrics and evaluation criteria laid out in the Initial Computer Design section above
        - The milestones that we achieved with the SolidWorks design was a schematic of the trap and a schematic of a single chip, both using simple polygons.
    - Chip circuitry design - This was not completed
        - The only metric for this milestone was whether or not we could fit the circuitry on the chip, and due to work elaborated on later in the report, this metric was effectively met.
- **Deliverable creation**
    - Rough drafts and final versions of all presentations
        - 491 Lightning Talk presentation
        - 491 End of Semester presentation
        - 492 Mid-Semester Update
        - 492 End of Semester presentation
    - Rough draft and final versions of both reports
        - 491 End of Semester report
        - 492 End of Semester report
    - Single (and subsequent if deemed necessary) takes of recordings
        - 492 Mid-Semester presentation recording
        - 492 End of Semester “How To” guide of the Digital Twin
    - A milestone we did not achieve was the rough draft of a paper outlining our accomplishments for publication in a research journal. This was assigned to one team member at the tail end of the 492 semester by our research advisor, who did not give adequate time or resources considering other deliverables needed for this course
    - Metrics and Evaluation Criteria for all deliverables are outlined in rubrics and culminated in grades given by advisors and judges of the CPRE 491 / 492 courses. These rubrics and grades will not be discussed in this report.

4 Design

The design is where the bulk of our work can be viewed. We start with looking at how our design fits into the broader context of quantum computing and the world in general. This has remained relatively unchanged since CPRE 491. We go on to discuss (briefly) the previous work and solutions in this field as well as the blatantly obvious technical complexity that accompanies designing a quantum computer. Part 4.2 dives into our design decisions and how we came up with / evaluated solutions. The final parts of this section dive into our specific design, any concerns and considerations regarding it, and our overall thoughts on what we’ve done.

4.1 Design Context

The design context dives into an overview of how the QC design affects various global-scale aspects, briefly touches on how it touches on and incorporates previous work done in the field, and highlights the technical complexities of the design and our project as a whole.

4.1.1 Broader Context

Area	**Description**
Public health, safety, and welfare	Due to the purely theoretical and research-oriented nature of our design, it has minimal immediate impacts on public health, safety and welfare of people. The design is intended for students and faculty in the field of quantum computing / quantum physics, and stands to further the goals of the field. Down the line, it is our hope that quantum computers will be commercially viable and unlock a new paradigm of computing power for the general welfare. We believe we are doing our part in furthering this goal. Hopefully, our contributions to the field of quantum computing will enhance public welfare, as prior advancements in technology have.
Global, cultural, and social	The most significant impact of this project will be in global, cultural, and social areas. With this being a design of Iowa State origin, although our findings will be publicly available, the project stands to benefit Iowa State oriented individuals the most. Iowa State is very behind in the race for quantum computing power and the end goal of this project is to reduce that gap through further recognition of the school's efforts, increases in staff count and caliber in this department, and student engagement in the field. Doing so will increase the longstanding culture at Iowa State of ingenuity. As a team, we say "If the first digital computer was invented at Iowa State, why can't the first large-scale Quantum Computer?" While ambitious, there is a chance of global effects of our design. If our design works and performs as we intend it to, this could help the global field of quantum computing move forward. It could help open a door for the QC community into another school of design.
Environmental	Again, due to the purely theoretical and research-oriented nature of this design, the environmental impacts of it are negligible at the moment. A physical quantum computer of this specification would require a great amount of energy to run, similar to other Quantum Computers (QCs) of today. The power requirements and necessity of running the computer at 10 Kelvin would be significant relative to many other senior design projects. Our QC would utilize materials standard in other QCs, but procedures around obtaining such materials may still be harmful to society and the environment. This is not something we have much affect over, as we would not be constructing the computer, and as such, sourcing the materials.
Economic	Should this QC stay within the realm of a hypothetical design, the economic impact will be very minimal. We hope that our design will spur further ideation and design creation, further leading to the financial viability of quantum computers as a whole. If we begin to construct a physical QC, the economic

	impact will be more substantial, due to the requirement of designing such a cutting-edge machine. We couldn't do any sort of component production at Iowa State and would have to outsource it to a DoE lab. Even still, our computer would most likely not be financially viable, and as such, not have any large macroeconomic effects for us or associated parties.

Table 1: Ramifications of our project

4.1.2 Prior Work/Solutions

The Honeywell and IonQ Quantum Computers have been built, but they have very few qubits, and are therefore limited. Our design is similar in many regards, but designed for a great many qubits, around 1,000. The design has achieved this by arranging multiple ion traps together into a node, and said nodes will be arranged to make a cluster. Additionally, our design uses a special memory-type ion trap that allows qubits to last longer.

Our design will be more like the Honeywell H1 QC. Both use Ytterbium (Yb) ions for qubits, each use electrodes to keep them in an ion trap, and each use beams of lights/lasers to set, address, cool and manipulate them. The Honeywell QC cools its Yb ions using Barium (Ba), while the IonQ QC does not. The Honeywell QC physically moves its qubits around to have them interact with each other, whereas the IonQ computer transfers information using light and swap gates.

Source: AVS Quantum Sci. 3, 044101 (2021); https://doi.org/10.1116/5.0065951

4.1.3 Technical Complexity

The complexity and scope of designing a quantum computer is apparent to us:

- The physics and mathematics of the operations inside a quantum computer are so complex (literally, they have a large focus on complex numbers, e.g. (a+bi,c+di)) that we have largely neglected them for the purposes of the design. We are treating ion-traps, a (relatively) extremely new and unproven design, to be a single, solid entity within our computer. Our computer will consist of a number of these ion traps in a specific layout to facilitate computing on a larger-scale than other QCs currently known in the public domain.
- Our problem was fundamentally an engineering problem - the layout and rough construction of a Quantum Computer with particular attributes. This holistically encompasses many aspects of the engineering design process - including ideation (see below), tradeoff consideration (technical and otherwise), and rough prototyping.
- Our Quantum Computer consists of ion-traps laid out in a novel design to accomplish basic computing responsibilities: mutilation and storage of data for multiple cycles. This is very similar to designing a traditional computer, but with an extra helping of mathematics and physics.
- The handoff of ions between traps relies heavily on particle physics
- The scope of our design problem is a kilo-qubit scale quantum computer - a novel concept yet to be successfully created. Designs for QCs with 10s or low hundreds of qubits do exist, but haven't been implemented due to cost, material, or technology constraints. We intend on making this computer with current technologies and materials, which will definitely be a challenge. Pushing the boundaries of a new field with existing technology will prove to be a sufficient challenge.

The complexity and scope of designing a digital twin to a quantum computer:

- **Why a digital twin is necessary:**
- A digital twin is necessary to simulate our ideas without having to create new quantum hardware for every new idea or test.
- We will need to use our quantum computer in the future instead of the qiskit backend.
- It is necessary to use some sort of quantum backend simulation until we have our quantum computer.
- We must have a mastery of our libraries and code to create a digital twin.
- **The complexities of creating a digital twin:**
- There is a lack of libraries for what we are trying to create, there are no libraries for quantum scheduling or node level design, so we are having to create all of our own code for this.
- There is great complexity to many of our ideas such as quasi-crystal memory traps
- We were not given much time to implement many of the complex ideas
- Understanding what we are creating and how to create it can be difficult, taking a relatively large amount of time to get the correct resources and ask the right questions to our clients

4.2 Design Exploration

The Design Exploration section highlights the thought processes and design decisions that we made across both semesters of senior design.

4.2.1 Design Decisions

In the Design Decisions section, we look at each of the questions we had to address over the course of both semesters. As time passed, our decisions became more granular and specific, and many decisions we had to make were based on larger decisions made earlier in time.

4.2.1.1 Design Decisions in CPRE 491

Number of clusters in the computer

- These are the largest layer of the QC. Depending on their role, the number of these may be important to not bottleneck the computer during operation. An adequate number of these is important to the "scale" aspect of our design, as multiplying these elements will quickly get our computer to the size we'd like it to be.

Number of nodes in a cluster

- These are the intermediate layer of the QC. Depending on their role, the number of these may be important to not bottleneck the computer during operation. The number of each of these in each computing node will set the upper limit of computations the computer could perform.

Number of traps per node

- These are the base layer of the QC. They serve a fundamentally different role than the larger two layers due to their importance to the physical operations of the computer. The number of each of these in each computing node will set the upper limit of computations each node unit could perform.

Function(s) of traps, nodes and/or clusters

- The decision to use all computing components as duplicates which do any one or number of functions would sway the number of each type of component needed to effectively perform computing operations.

Physical orientation of traps relative to other traps, nodes to other nodes, clusters to other clusters

- The orientation of each component is of paramount importance with respect to the physical tradeoff of ions, ability to hold information for multiple machine cycles, and ability to use quantum computing effectively.

4.2.1.2 Updates on Design Decisions in CPRE 492

In 492, the larger QC design team split into the Hardware and Software sections. Each team ran with different aspects of the design, and as such, encountered different challenges and had to make different decisions which largely didn't affect the other half of the team.

Decisions the hardware team made during 492:

Number and size of ion traps

- While the number of ion traps per node was something that was discussed and ultimately decided upon during 491, our team had to decide if that number was reasonable to fit onto a single chip that would be able to function.

Ion trap orientation

- While we made very small strides in addressing this problem in 491, a bulk of the thinking behind the manner in which the ion traps would be situated was done this semester. This was also largely affected by the number and size of ion traps on a chip.

Laser operations on ion traps

- In a similar vein to ion trap orientation, the exact operations that needed to be performed by lasers to ion traps / ions was a big question going into this semester. We had to figure out how many lasers would be used and along which plane would the lasers address their respective target.

Number of, size, and type of ancillary hardware needed for laser operations

- The laser operations being performed had to be accompanied by specific physical hardware components that are discussed in later sections. We assumed that such components may have been necessary for proper operation when we were entering the semester, but did not know for sure.

Orientation of laser producing modules and ancillary hardware

- Lastly, based on basically all previous design decisions we had to make, we had to tackle arguably the hardest question: Could we fit the laser producing module and ancillary hardware in / around the quantum computer in a feasible manner. The primary driver of this decision was whether or not the lasers / hardware could "see" what it needed to for proper computation.

Decisions the software team made during 492:

What software to create first:

- Deciding what software to work on first is very important due to the nature of not having much time to work. We created the fundamentally necessary classes and functions in order to have as much done as possible. There were more minor and complex things we could have implemented such as the quasi-crystal memory qubits. We chose not to implement

the memory qubits because it would have been very complicated and we wouldn't have been able to implement the scheduling or have as good of documentation.

What language to use:

- We started a program in C++ but we moved over to a python program in order to take advantage of the useful libraries. Qiskit is a python library and allows us to easily run simulations of our quantum circuits without having to program it ourselves.

How to structure the software architecture:

- We decided to base our architecture on the idea that we would be solving a scheduling problem. We created our classes in a way that allows us to have a class specifically dedicated to optimizing the scheduling of operations on qubits. Having separate classes based on different things allows for our code to be more easily understood and altered in the future.

How to schedule the operations

- There main concepts that the software team focused on are decoherence and noise. We tried to decrease the decoherence by using qubits as late as possible. This is counterintuitive to classical computing but qubits cannot be stored for long periods of time, meaning we want to use them as late as possible. When qubits are moved, there is noise involved and so ideally you want to move qubits next to each other, then start the operations. We want to limit decoherence and noise because they decrease the fidelity of the computer, which is essentially how accurate the computation was. If fidelity is low, computations will fail.

4.2.2. Ideation

The Ideation section outlines how we contemplated different solutions to a problem. In our fall semester (CPRE 491), we only dealt with one issue. We spent multiple weeks thinking of different potential solutions and evaluating each option before moving forward. In our spring semester (CPRE 492), each team dealt with multiple issues and quickly iterated through each problem with our first proposed solution.

4.2.2.1 Ideation in CPRE 491

During the first semester, we had to decide upon the physical orientation of traps relative to others. As previously mentioned, the orientation of the ion traps relative to other ones is fundamental in the handoff of ions between traps, a crucial standpoint of our design and the driving force behind our scalable, modular design.

- **Square Grid design with upper and lower tracks**
    - This was the design recommended by the advisors of our project. This type of design is supported by modern, 2-level wafer electronics printing and was what we ended up moving forward with. Similar to a street-level grid used in many newer towns, a meeting point would consist of two, three, or four “roads” (ion traps) converging in “intersections.” This design calls for explicit usage of the cardinal directions to maintain order
- **Tree design**
    - Two major factors permeate our design requirements. Firstly, the addition of ion traps at their intersections introduces additional noise and uncertainty, as a qubit traveling from one to another is less likely to go in the intended direction.

Secondly, the movement of a qubit reduces its coherence time, and introduces error into the QC. Therefore, we would do good to minimize both the number of ion traps at each junction, and the distance between each qubit. A tree-like design, such as a binary tree, compromises between these constraints. It allows for each qubit to only need to travel O(log(n)) ion traps to get to any other qubit,as opposed to a line or circle, which requires O(n) ion traps to be traversed. It also requires less ion traps per junction between ion traps than a grid or wheel spoke. Even if the design is not strictly a tree, it can have aspects that are like one.

- **Wheel and spoke design**
    - This was a design thought of by a team member. It involves many ion traps converging at a central point, where ions could be handed off to any one of the number of "spoke" ion traps. Around the outside of these spoke traps, we could have a "wheel" of ion traps providing a potentially different function.
- **Triangular grid design with upper and lower tracks**
    - This is a slightly different iteration of the grid design. Instead of having squares, we could have a triangular grid, with each connection being a meeting point of three ion traps instead of two or four. This design would not use the cardinal directions, and all intersections would have exactly three connections (outside of those on the corners of the grid).
- **Other 3+ Layer designs**
    - This is a subset of thoughts that we came up with when considering the binary tree design. This school of design requires the practical capability of electronics manufacturing with 3+ layer wafer design. We did not look much into this option, as to our knowledge, such wafer design is not possible.

4.2.2.2 Updates on Ideation in CPRE 492

As mentioned in 4.2.1.2, the Hardware and Software sections of the design diverged in 492. Each team ran with different aspects of the design, and as such, encountered different challenges and had different ideas on how to overcome them. We dealt with multiple separate problems and usually did not come up with multiple options for each decision, but ran with our original choice unless we needed to redesign it. We also had to be slightly more conscious of how our ideas may affect the other team, which we briefly address in the tradeoffs section.

Hardware Decisions:

Number and size of ion traps

- For 492, we continued to run with our previous idea for number of traps per chip (specifics in 4.3)
- For the size of the ion traps, we initially passed our design off of the HOA (High Optical Access) trap's measurements. This is the only public ion trap design to our knowledge.
- Near the end of 492, we were informed the HOA trap would not be able to perform the ion handoff due to how the electrodes were set up. With no other publicly available trap designs, we were forced to move forward with the HOA's measurements as a "proxy' for what an actual trap should be, size-wise.

Ion trap orientation

- when we were thinking about how to arrange / orient the ion traps on the chipo, we went for a layout that maximized the distance between any two ion traps regardless of if they were on the same chip (ie the above chip or the below chip)
- A big deciding factor was how to orient the traps so that they could be addressed by various lasers. having some part of the trap near the edge of the chip and at a right angle relative to the chip was very important.

Laser operations on ion traps

- There wasn't as much traditional "ideation" involved in this decision, but moreso referencing proper papers and consulting with our advisors about what needed to be done to get the computer ready for / actually doing the computation. Our PhD student, Gavin Nop, was particularly helpful here.

Number of, size, and type of ancillary hardware needed for laser operations

- Similar to the previous, referencing the proper academic resources was how we came up with our 'solution' to this challenge.
- Our solution for this is much less specific than any other challenge. This is because no paper goes in depth into the specifics of the ancillary hardware, and all of our advisors have never seen a real quantum computer (as such, don't know the ancillary hardware we would need). We were able to piece together an idea of what we needed / roughly how many of each thing we would need, but not the make and/or model of the ancillary hardware used in similar quantum computers.

Orientation of laser producing modules and ancillary hardware

- This was the truest "ideation" process we had to undertake. This was the last design challenge we worked on, and we don't have a great solution due to how late we undertook this. The requirements were pretty open-ended, so much so that there were various solutions that we didn't even think of initially.

Software Decisions:

What software to create:

- There was not much change in ideation of what software would be as it was only mentioned a few times in 491. The nature of this project is that we are creating things without much guidance due to it being very "ground level". The graduate student associated with our project in 491 created a physics simulation for keeping qubits in ion traps and executing static handoffs. We decided to not work further along this due to the expertise required. When considering our client's broad goal: a digital twin that addressed any and all implementation concerns possible, we knew we had to limit the scope. Our software team decided to solely focus on the digital twin as a compiler and "executor" of quantum algorithms respective to our model of the hardware. It has many benefits, as it allows us to simulate the high-level concepts that are unique to our design, and it can be extended to greater depth and interfaced for use with the hardware in the future.

Scheduling Policy:

- By the time our team had a clear view on what the issue our client wanted us to solve was, we had limited time, and many implementation options available. To create a true optimization for the scheduling of quantum operations would have required both the

acquisition and application of a substantial amount of technical knowledge: more than we had procured in the previous semester. As such, we identified 3 possible policies that attempted a very low-granularity treatment of our quantum-computational concerns: as-late-as-possible, as-close-as-possible, and as-fast-as-possible. Barring some lengthy details of each implementation: our team collectively discussed and ran through a shortlist of hypothetical small-algorithm executions under each paradigm, and after consultation with our client, determined that the as-late-as-possible paradigm would yield the highest performance on average, and be most relevant to more optimal scheduling policies that could later be implemented as extensions/modifications of the ALAP policy.

4.2.3 Decision-Making and Trade-Off

The methods in which we evaluated each solution differed greatly between semesters due to the number and variety of problems we faced. We faced a singly, impactful decision during CPRE 491, leading to us contemplating multiple designs thoroughly. During the spring semester, we placed less emphasis on ideation and more emphasis on speed, leading to less trade-off analysis and overall less time spent in the decision making phase.

4.2.3.1 Decision-Making and Trade-Off in CPRE 491

Physical Layout of computer

- **Grid design with upper and lower tracks**
  - Pros:
    - Moderate junction density, at most 4 for a square grid
  - Cons:
    - Moderate-High O(n^(1/2)) distance between qubits
- **Tree design**
  - Pros:
    - Moderate-low O(logn) distance between qubits
    - Moderate-low junction density, at most 3 in the case of a binary tree.
    - Principles of the tree design may be applied to other designs.
  - Cons:
    - Leafs have the worst travel time, and they are the most prevalent.
    - Other designs have better qubits distance or junction density individually
- **Wheel and spoke design**
  - Pros:
    - Low distance between qubits
    - Low junction density on the wheel, 2 or 3 ion traps per junction.
    - Could support parallel computing in multiple “spoke” ion traps
  - Cons:
    - High junction density, particularly in the center of the wheel
    - Ions could get lost in the hub of the wheel, where the spokes all meet
- **Triangular grid design**
  - Pros:
    - Moderate-Low distance between qubits.
    - The structure would be most dense, and take up less space.
  - Cons:

        - High junction density, at most 6 for a triangular grid
- **Other 3+ Layer Designs**
    - Pros:
        - Denser, taking up less space.
        - Vertical traversal allows for shorter qubit traversal time.
    - Cons:
        - Likely high junction density
        - Probably not possible with current technology

We have not yet made an official decision on this or other considerations yet, as knowledge acquisition is still underway to help more thoroughly inform our decisions.

4.2.3.2 Decision-Making and Trade-Off in CPRE 492

In 492, we placed less emphasis on ideation, and thus, less "tradeoff analysis" was done. We placed more emphasis on workable solutions due to the larger number of decisions that needed to be made. Most of the time, we would move forward with our initial design and only come back if there was a critical flaw.

The hardware side only had two scenarios where we had to decide between 2+ workable solutions:
Ion trap orientation

- Our initial design called for not having the wirebond section of the ion traps close to each other regardless of plane. We did this because we believed this would minimize the chance of any of the 30+ wire bonds on each trap from getting tangled with each other.

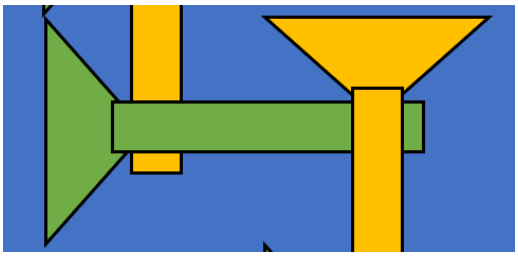

Figure 1: Example of initial ion trap orientation

- Later on, we decided that a more uniform design with all wire bond sections located as close to the edge of the chip was preferable. We came up with this by thinking on a more global scale, and it turned out having most of the wires and circuitry located around the outside of the chip made it easier for circuit design and meant we could get significantly outsized returns to cleanliness by making the chip just a little bigger.

Orientation of laser producing modules and ancillary hardware
Designed that were considered included:

- **Hardware located in-plane and close to chip**
    - Scrapped because ancillary hardware was deemed too large to fit close to the chip
- **Hardware located in-plane and far away from chip**
    - This was (and still is) a viable design for our computer, but would be slightly harder to work on set-up wise due to the still relative closeness of our ancillary hardware compared to our design we selected

    - The fact that everything is on the same plane is both the pro and the con of this option, as it makes it slightly easier to work on but means that there is slightly more congestion and a higher chance of unintended overlap.
- **Hardware located out-of-plane and far away from chip**
    - This is our current solution. We discovered that it doesn't really matter where the ancillary hardware is on the X, Y, or Z plane, so long as the lasers nearby the ancillary hardware could be directed to a right angle relative to the chip via mirrors. The ancillary hardware could be arbitrarily large so long as the laser lengths emanating from our ancillary hardware / laser producing cluster were arbitrarily long.
    - As Nick put it "The ancillary hardware could be the size of a building if each laser were just 5 miles long."
    - The pro of this design is the aforementioned arbitrary size limit on ancillary hardware, while the con is that ancillary hardware may be located non-optimally for lab work.
- **Non-ion-trap software backend**
    - The quantum computer that our backend runs on is not an ion-trap. It would have cost money in order to get the ability to use IonQ, our only option. We decided it wasn't worth the cost to have a more realistic backend. One of the main factors is due to the fact that we are using the Qiskit library, we wouldn't be given more functionality and we wouldn't be able to move qubits or do our own scheduling. Overall we decided the slightly more realistic simulation would not be worth the cost.
- **Degree of Implemented Scheduling Policy Optimality:**
    - As discussed in a previous section, the requirements for implementing a fully robust quantum scheduler were beyond our capacity. In its place, we opted for a policy we referred to as "as-late-as-possible", which placed the highest priority on completing operations on a given qubit as close to its measurement as possible. This won out over policies that grouped operations utilizing the same qubit as close as possible while maintaining a conflict-serial schedule, and those which prioritized getting all qubits to their final state as quickly as possible. This was the result of a consultation with our client, from which we concluded that this would have the greatest impact on average. It was also relatively simple to implement, and will provide a more effective baseline for generating conflict-serializable schedules to be used by a more robust scheduling optimizer that more faithfully addresses quantum mechanical concerns in the future.

4.3 Proposed Design

The bulk of our project lives in the Proposed Design section. This section explains the design of our quantum computer, starting with an overview dating back to the end of CPRE 491 and then delving into how it works and modifications that were made during CPRE 492. We finish off by discussing areas of concern both past and present.

4.3.1 Overview

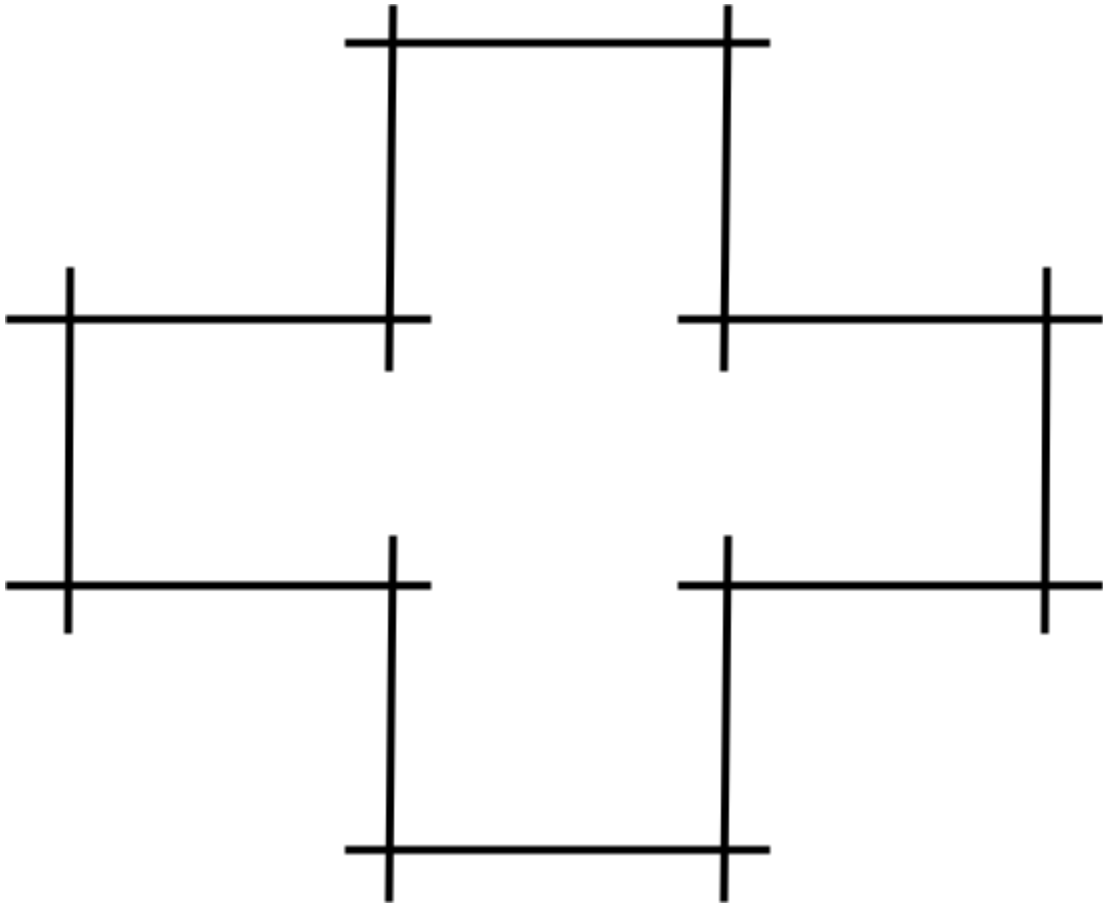

Figure 2: A brief overview of the shape of our RF ion trap design

We have designed a quantum computer schematic. Pictured above is our design for a node of this computer. This node will be composed of 12 ion traps, and each will hold ~10 ion. Each ion acts as a quantum bit ("qubit"). Each of these nodes operate in the same fashion, and connect to each other via a currently unknown method. This is the paradigm of a different Senior Design team and wasn't within the scope of our project.

To transfer information within a node, qubits continuously move between ion traps at junction points near their ends. Due to the state of current technology in the field, it is best to implement junctions between two ion traps with a right angle. The part of each ion trap that extends past this junction helps to guide the qubit to its destination. Additionally, the ends of each ion trap contain a DC voltage stop - a wall that stops ions from flying off the ion trap. The cross design is composed of only right angle junctions, and is, to some degree, symmetric, so forces from electrodes will cancel each other out, which is desired.

4.3.2 Detailed Design and Visual(s)

We have designed a quantum computer schematic. The central part of the computer is a node, which itself is a quantum computer holding a number of qubits on the order of ~100. The node will be used in clusters to build a quantum computer of many clusters, which in total can hold on the order of 1000 qubits.

A "qubit" is a quantum bit in a superposition between 0 and 1. A single qubit may be represented as a vector of two complex numbers, the magnitudes of each are the probabilities that the qubit will be either 0 or 1 once it is measured, so the magnitude of the magnitudes of the elements of the vector is always 1. Qubits may be manipulated by quantum gates, which may be thought of as unary matrices, that is, any matrix when multiplied to its conjugate transpose becomes the identity matrix. This allows the probabilities of a qubit being 0 or 1 to be flipped, or for the polarity of one of the aforementioned complex numbers to be flipped, or for a qubit to gain an equal probability of yielding 0 or 1, from a state of exactly 0 or 1, but do nothing when applied twice. There are also two-qubit gates, such as the controlled not gate, which flips one qubit's value if and only if the other will yield 1 when measured. This allows for quantum entanglement, a situation where one qubit will not yield a different result than another qubit will when the same outside stimuli are applied to both. Complex quantum algorithms take advantage of these qualities to perform tasks faster than on digital computers. Though the complex numbers cannot be measured directly, their magnitudes can be measured by running the same algorithms multiple times, and measuring average responses.

As stated prior, qubit information is stored in Ytterbium cations, which are called "physical qubits." The last valence electron in the Ytterbium cation exists in a superposition between its usual orbital, and an excited state, allowing for us to use the ion (electron) as a qubit. Multiple physical qubits may be used together to model a single qubit with greater accuracy, called a logical qubit. We use lasers to emit photons, which then impart the Ytterbium atom with energy depending on the wavelength, as the last electron can absorb the incoming light, and jump to a higher state. Depending on the frequency of light provided, the electron could jump to a specific unstable state, and then go back down to its usual position. This allows us to set specific qubits to be either 0 or 1. Furthermore, we can measure the orbit of the electron by sending light to it, and then measuring what we get, because some light may be absorbed by the ion and will not go through it. Using these lasers, we can set and access the qubits.

In order for the qubits to be useful, the ions must exist at a very low temperature; otherwise, there will be too much noise, and the physical qubits we do have will retain information for less time. This temperature must be around 10 - 12 Kelvin for the computer to operate. Therefore, we will employ multiple methods of cooling, including doppler cooling and passive Barium cooling. In doppler cooling, we emit light at a specific frequency to cause the Ytterbium ion to lose energy and slow down. In passive Barium cooling, we place Barium ions between the Ytterbium ions, and then cool the barium ions more aggressively, which then cool the Ytterbium ions. The image below depicts an ion gate with ions, and a beam of light going from the top to the bottom.

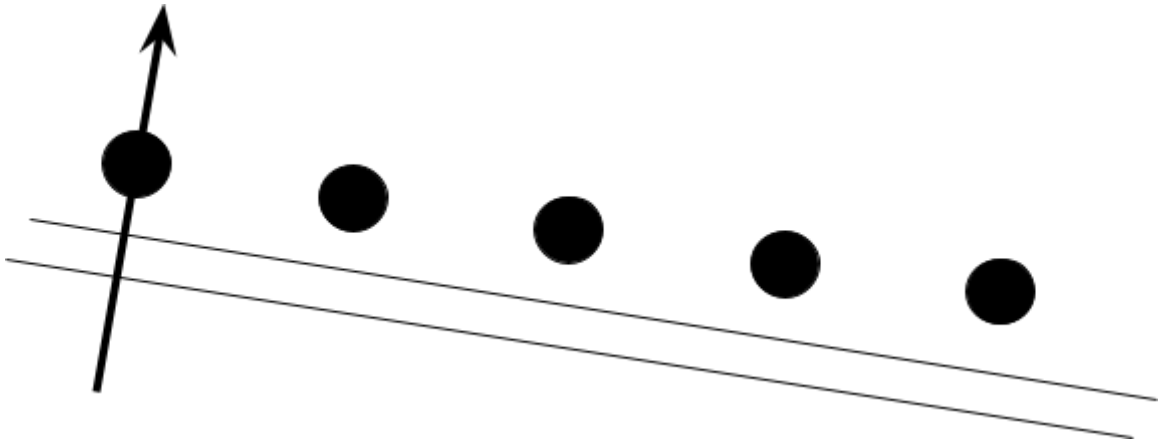

Figure 3: Ions suspended above an ion trap

Useful quantum algorithms are complex, and require many qubits to execute. Therefore, our machine utilizes many qubits, on the order of 1000, to advance the field of quantum computing. The primary challenge to accomplishing this feat is that qubits are very unstable, and exist for only a short amount of time. Usually, the addition of more qubits in a quantum computer introduces noise, and reduces the amount of time a qubit can hold accurate information. Therefore, we use special memory-specific ion traps that are designed to only store qubits without doing gate operations on them. This is a key point where our design diverges from existing designs.

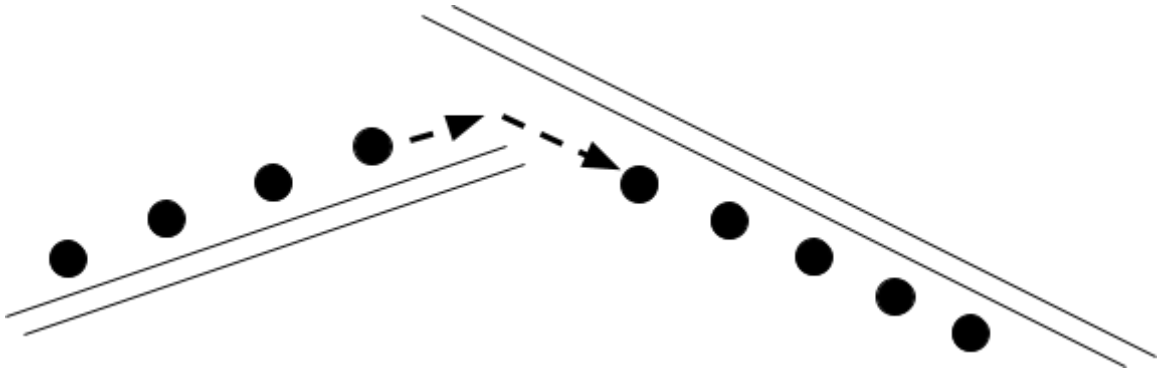

Figure 4: Visualization of an ion handoff between two ion traps

In addition to using memory specific ion traps, we also use a vertical transfer, shown above. In this, qubits will transfer from one ion trap to another vertically, just hovering across. We expect that this will limit the noise that a qubit encounters while traversing ion traps.

4.3.3 Updates to Detailed Design and Visual(s)

In 492 we updated the design on the hardware front and essentially started the design on the software front. Challenges and ideas for each element of our design can be found in previous sections, as 4.3.3 will only cover the product of our labors. The hardware portion will have substantial visual references in this section, whereas software will be heavier on the explanations. The code used is at the bottom of this report.

Hardware Design:

Ion trap size

- The ion trap measurements we went with were exactly those of the HOA trap. As previously mentioned, these are being used as a proxy due to the HOA trap's inapplicability to our specific design.
- We also discovered that components called the "Spacer" and "Interposer" need to be placed below the actual ion trap for proper functioning.

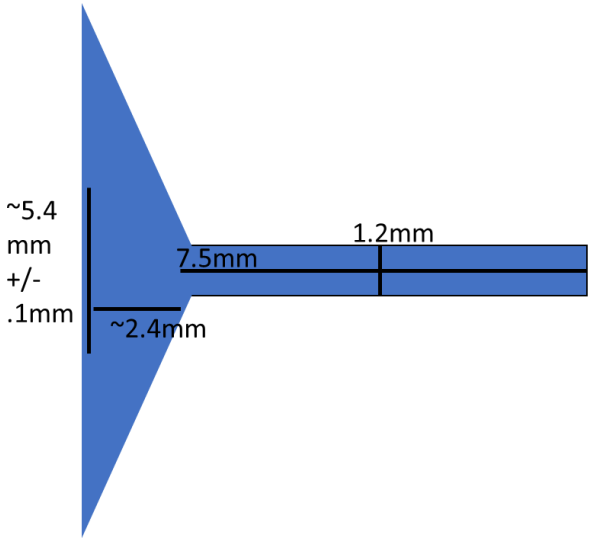

Figure 5: A measurement-based schematic of a single HOA ion trap

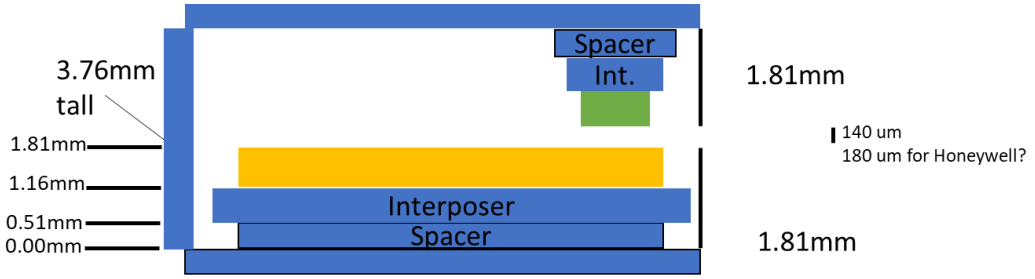

Figure 6: Side profile of the quantum computer. Purpose of yellow and green traps addressed in following section

Ion trap orientation

- Again, this was largely decided upon prior to 492. We moved forward with our cross based design. In 492, we also updated the design by standardizing the location of our RF addressal pads. Now, each addressal pad is facing outwards from the center, towards the edge of the chip. This makes it easier to implement the wiring necessary for this circuit. We also elongated each track, just so every component is able to fit on our wafer. This was one of the main takeaways from the Solidworks design.
- In the following image, the green traps represent those on the upper wafer, and are memory traps. The yellow traps represent those on the lower wafer, and are computational traps.
- Based on the sizes of the ion traps decided upon in the previous step, we decided a chip size of 26mm would be ample.

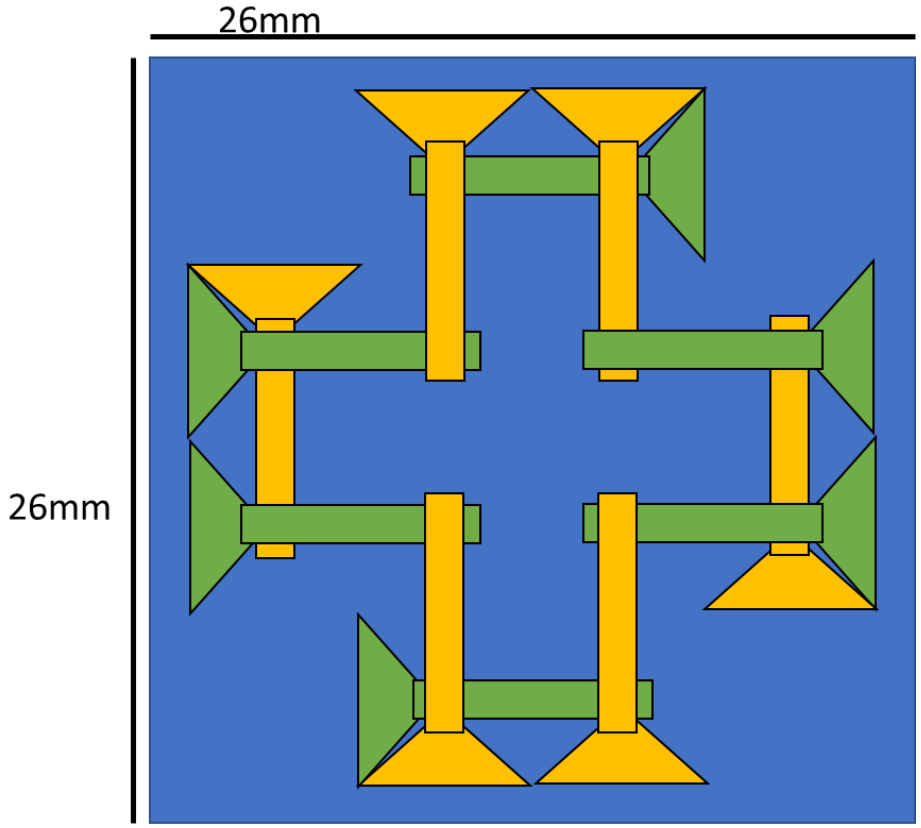

Figure 7: A top-down view of the central quantum computer, with top chip removed for easy viewing

Laser operations on ion traps

- Using academic resources and professional guidance, we deduced we would require the following lasers with the following purposes:
    - Two Barium Doppler Cooling Lasers, one with a 649.9nm and the other with a 493.5nm wavelength
        - These lasers will be prevalent during initialization and intermittently

    - Cooling lasers are not used at the same time as actual computation
- Two Yb ionizing lasers: 369.5nm & 398.9nm wavelengths
    - These lasers are used once per trap to ionize the Yb atoms, used before any other lasers
    - The 369.5nm laser is also used for readout of ions
- Two Yb Optical Pumping lasers are needed, with 369.5nm & 935.2nm wavelengths
    - These are used intermittently through computation to reset ion values
- One Addressal laser of unknown wavelength
    - This laser will be shot "at the side" of the ion trap, in a manner that it can view the entire region of the trap that holds ions. This contrasts with all other listed lasers, which will be shot "down" the ion trap.

Number of, size, and type of ancillary hardware needed for laser operations

- Some operations performed before / during / after computation require lasers to be fire through pieces of hardware. The main two piece of hardware are Acousto-Optic Modulators (AOMs) and Electro-Optical Modulators (EOMs). The purpose of these pieces of hardware are not discussed in this report and weren't particularly relevant for us,
- A serious challenge was the non-specificity of the exact type of each piece of hardware needed. AOMs and EOMs come in all sorts of shapes and sizes, and no one knew what differentiated one type from another on a basis relevant to our project, even our professional guides.
- The 493.5nm Barium Doppler Cooling laser is run through a sequence of two AOMs
- Both Yb Optical Pumping Lasers are run through separate EOMs
- The Global Addressal laser will need to be run through some sort of diffractor to be able to reach all areas of the ion trap, as well as an AOM
- All measurements of AOMs and EOM were used as aggregates of sizes pulled from datasheets of various components. These dimensions are more ballpark figures than in reference to any specific piece of hardware.

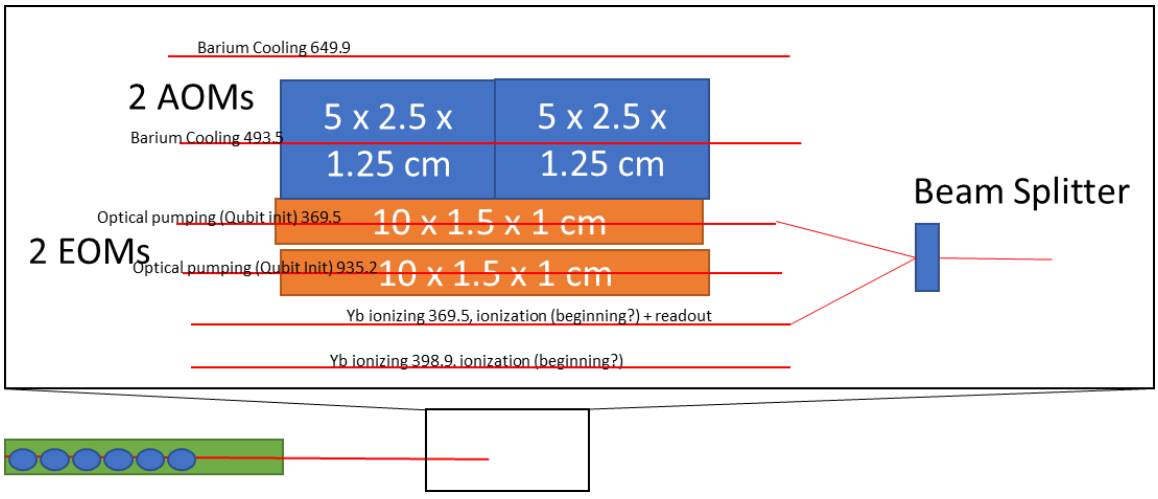

Figure 8: A cutaway view of all lasers and ancillary components that will need to feed into the ion trap

Orientation of laser producing modules and ancillary hardware

- The last issue was tackled was where to position the lasers and hardware. We decided to use reflectors and position all ancillary components arbitrarily far away.
- Depending on which direction the RF addressal pads of the ion trap face, and which chip the ion traps are on (upper or lower), we positioned the reflectors to point upwards or downwards from the chips' plane.
- Yellow squares are meant to reflect the lasers from a position above the chips, whereas orange squares are meant to reject the lasers from a position below the chips .

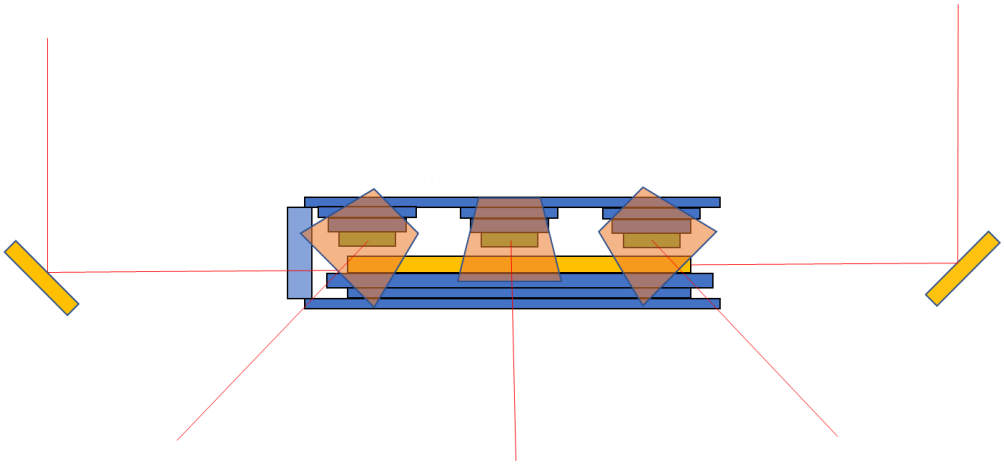

Figure 9: Side profile of reflectors and lasers.

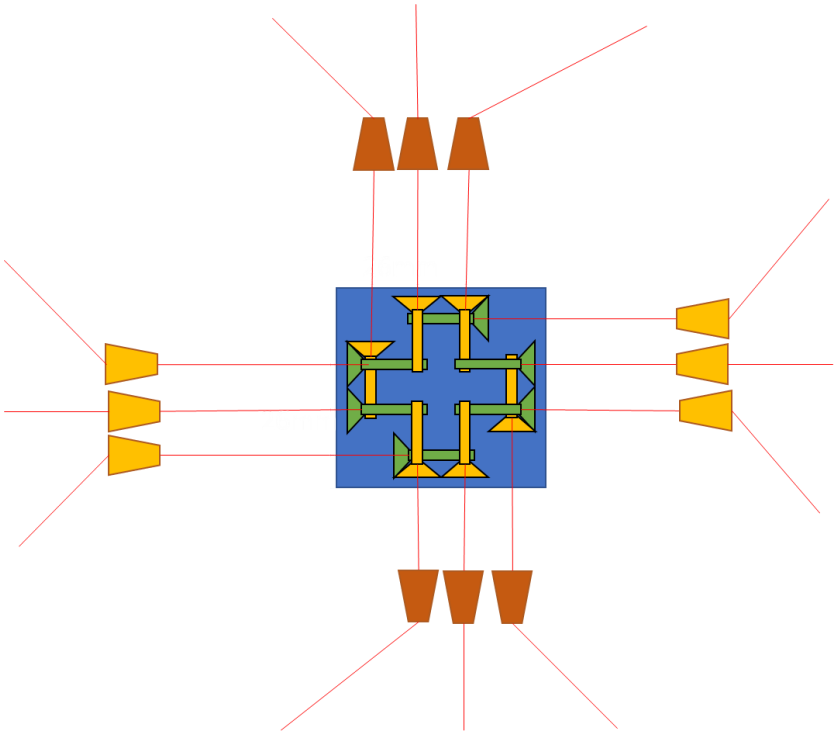

Figure 10: Top-down view of reflectors and lasers. Top chip removed in graphic for better viewing.

Software Design:

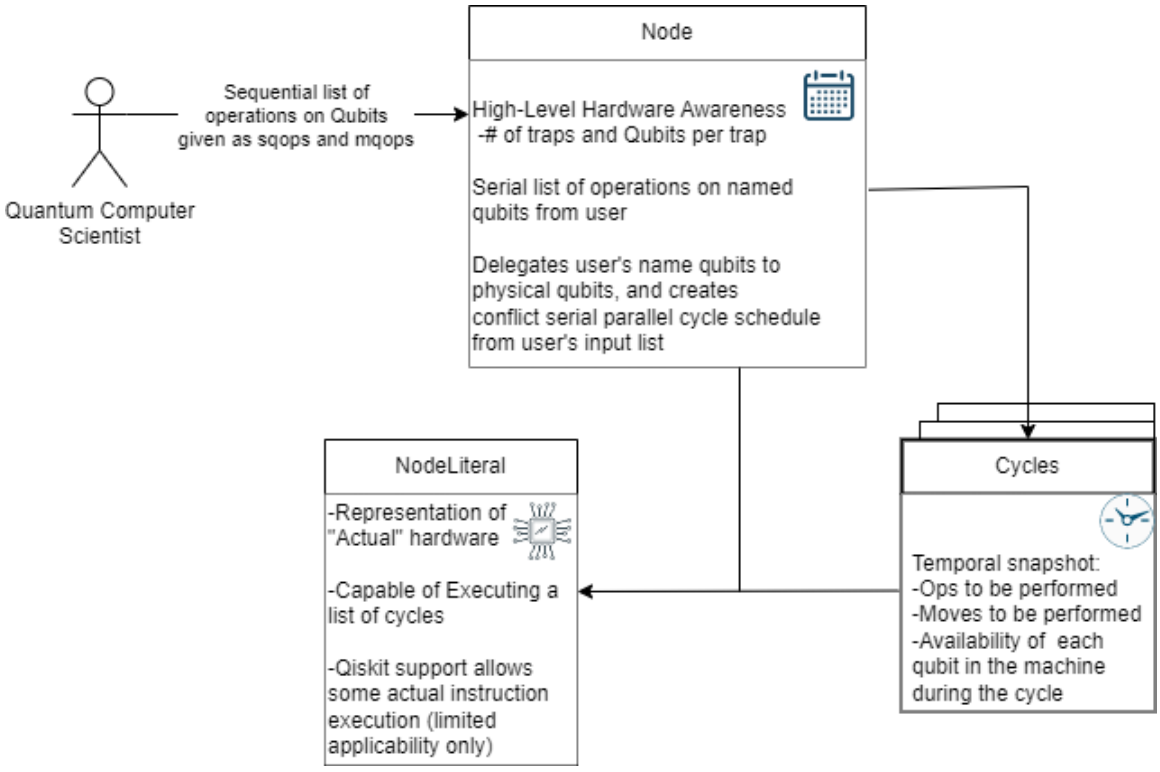

Figure 11: UML Diagram of Digital Twin

Overview

Functionality:

- We optimize execution of a quantum circuit by:
  - Simulating circuit execution in our hardware design
  - Taking advantage of parallel processing
  - Accounting for necessary Qubit movements
  - Integrating with Qiskit backends

Target User:

- Future project developers/maintainers
  - Developed as a library, our software is supposed to be a backend
- Users specifically trained in how to use our software

- We do not need to make our code accessible to people who have no knowledge of our project. This allows us to not create UI's and use technical language throughout our project.

Substance - Three Classes:

- **Node class:** The Node class is given a sequence of instructions and computes a dependency mapping between operations and resources. It constructs cycles to be used in the NodeLiteral class.
  - Node(n): Constructor
  - sop(q,op,ph): Single-qubit operation
    - Supported gates: x, y, z, h, p
  - mop(p,q): Multi-qubit operation (C-NOT Gate)
  - describe(): Prints a description of the node
  - getLiteral(): Generates a NodeLiteral from the current Node object
- **Cycle class:** The cycle class simulates a cycle of a quantum computer cycle
  - Cycle(n): Constructor
  - sop(p,op, ph): Single-qubit operation
  - mop(p,q): Multi-qubit operation (C-NOT gate)
  - mvq(p,q): Qubit movement: Swaps two adjacent qubits
  - describe(): Prints a description of the Cycle to stdout
- **NodeLiteral class:** The NodeLiteral class simulates the hardware level node, it directly interacts with qubits, there are no optimizations or controls here
  - NodeLiteral(n): Constructor
  - addCycle(cycle): Adds a Cycle: a list of operations to do in a machine cycle
  - describe(): Prints a description of the NodeLiteral to stdout
  - setOrder(order)/getOrder(): Assigns/returns logical qubits to initial locations
  - getcirc(): Generates and returns a Qiskit QuantumCircuit
  - execute(backend): Executes on a Qiskit backend

```
# Make A Node
n = Node(8)
n.sop(0,'x')
n.sop(1,'y')
n.sop(0,'z')
n.sop(2,'x')
n.sop(2,'z')
n.mop(0,3)
n.sop(0,'p',.5)

# Print Description To Validate
print("NODE DESCRIPTION")
n.describe()

# Generate NodeLiteral
l = n.getLiteral()

# Print Description To Validate
print("NODELITERAL DESCRIPTION")
l.describe()

# Generate Circuit To Validate
circ = l.getcirc()
print(circ.draw('text'))

# Generate And Execute A Circuit For Validation
backend = BasicAer.get_backend('qasm_simulator')
r = l.execute(backend)
print(r.get_counts())
print("END PROGRAM")
```

Figure 12: Software example user code

4.3.4 Past / Present Areas of Concern

The silver bullet of our design lies in its nature - it is simply a design. Until a real computer that's based on the hardware of this design is constructed, we will never know if our design works. To that end, some of the most important factors of our design thus far may not even work, which we refer to later in 4.5 Design Analysis. Further development on our design will be largely dependent on access to high-grade electro-dynamic simulations, a probable precursor to building the actual computer. With our existing simulation software, we can not be sure that the results of our simulations accurately represent a real world outcome. We must be certain that under scrutiny from more well-endowed individuals and institutions with access to more advanced modeling software, our design holds up.

Hardware

On the hardware side, we have broad areas of concern around the feasibility of production of this design. This was something that we were concerned with at the outset of 491, but as we've made design decisions and had to "hand-wave" some of the details of this design away due to lack of

publicly available knowledge, these concerns have become ever more present. Specifically, the fact that we don't have an exact ion trap to base this design off of is cause for great concern. Knowing that the single public design for an ion trap can't be applied to our design means that the exact dimensions of a viable ion trap are up in the air. Additionally, there are minor concerns around ancillary hardware. No one on our team nor none of our advisors have seen a quantum computer, so the size and number of ancillary hardware was backed into based on tangential writings of people who have seen a QC "in the wild." We think that our design more or less solves this problem by locating ancillary hardware some distance away from the trap, but until a computer based on our actual design is constructed, we just don't know.

Software

There are many concerns with creating a digital twin to a quantum computer. We need our code to be compatible with other teams. There is a team creating a cluster level digital twin and if we create incompatible code then it would be a major setback in the future. Software is only as useful as it is in the real world. There are reasons why companies choose to create slower code that is more abstracted. It is incredibly important to have code that can be expanded upon and is understandable by others. If our code cannot be understood then it will become completely useless as soon as someone tries to expand upon it. Our code must be accurate in testing what our hardware team wants to implement otherwise we may test one thing and end up with inaccurate results.

4.4 Technology Considerations

Our design for a quantum computer is made out of one modular component - a Ytterbium ion-trap. While a fairly recent development within the broader scope of computing, this is a tested way of performing computational operations. These ion traps have two segmented RF electrodes with DC voltage steps that create a "tube" above the trap that the ion sits in.

While our initial design used the High Optical Access (HOA) trap outlined in a 2016 paper from Sandia Labs, we discovered at the end of the semester that this ion trap does not support the kind of "ion handoff" that is a fundamental part of this computer's operation. Designing a new ion trap is well outside of the scope of this class, so we moved forward using the HOA as a basis for measurements of this design.

We initially started and still continue to base our design off of a Ytterbium ion-trap design because the current existing king, the Honeywell H1, has been continuously breaking records in terms of quantum volume. Quantum volume is a metric introduced by IBM in 2019 to show the computational power of a quantum computer. The H1 has set the last three records of quantum volume, and has 10x'd their quantum volume annually. Other types of quantum computers, such as those utilizing superconducting ions or using material defects to trap ions, are either too unsophisticated at this point in time or have had their recent progress slow.

The last element of our hardware design is the usage of Acousto-Optic Modulators (AOMs) and Electro-Optical Modulators (EOMs) and accompanying beam reflectors. These components are used for initialization cooling, and ion trap computations. We did not explore in-depth these facets of our design, as they are better suited to people with more lab experience. We believe our design will not be novel in this sense whatsoever.

Our software technology must be able to run quantum simulations in a considerable amount of time. Scaling up to the level of the actual quantum computer could be very intensive and we will primarily have a proof of concept. It is worth considering that different languages can be used for different parts, such as python being useful for the qiskit library.

One of the most important parts of our project is that we will use existing technology, materials and methodologies. As alluded to, we are not in the place to create new and advanced materials or technologies, financially and educationally. The pro of doing this is that we don't have to invent something new, but the con is that utilizing existing technologies to come up with a design that outperforms existing ones in many respects will be a challenge, which we have now experienced first-hand. We still believe that the continuous transference of the ion between traps could be a game changer, if it works.

4.5 Updated Design Analysis

We have not built or implemented any hardware components. Due to the fact that our project simply calls for a non-physical design, there will be no construction or physical implementation of the design. We currently believe continuous exchange of ions between traps to be possible, and our overlapping-ion-trap design hinges on it being so. For this reason, there are serious implications for the overall feasibility of our design.

Though we have not built any hardware components, a digital twin allows us to simulate and test these components without the high cost of building them. The digital twin will be used in tandem with the quantum computer once it's built in a few ways. It will be able to send instructions for moving and scheduling qubits. The digital twin will also be able to send the gate operations that are to take place.

As previously mentioned, the difficult part of this assignment was the creation of the design. There is a high likelihood that in the end that some of our subsystems or entire design may not 1. Be functional 2. Push the boundaries of quantum computing or 3. Be feasible with current technology. Put another way, we are not only working in the paradigm of "Will this be good or change the game?" but also "Is this even possible?"

5 Testing and Security Concerns

Part five of our report covers testing. Seeing as our design is not physical, we didn't really do any physical testing. Any sort of physical testing will be performed by individuals in the future, if / once the design is manufactured. The software team would like to have formal testing done as a next step but unfortunately we did not have enough time to implement any testing except for one manual test function. We were only very recently told that we would even have to create a digital twin, though given more time testing would be our next step.

5.1 Hardware Testing

For the project, as mentioned, nothing physical was constructed. During the hardware design process, testing was done during construction in SolidWorks and through rudimentary circuit design. We actually ran into a circumstance in 492 where the approximate sizes of ion traps drawn in a PowerPoint weren't to scale. When we tried to construct the SolidWorks design with ion traps of actual (HOA) trap measurements, it didn't look right. This 'pseudo-testing" was crucial in our hardware design, as it revealed problems with ion traps / other components physically running into each other or overlapping each other such that ion traps couldn't be properly addressed. The circuit design aspect was more of an afterthought, simply ensuring that we make the chip large enough so that we could hook up all ion trips to the chip's wiring.

5.3 Software Testing Process

Our process would have been to create unit tests to test individual components. These tests would be run whenever someone makes an update to the code to maintain code quality. We would want to test edge cases as well as normal use cases. We would want to create tests for individual components to make our code as flexible as possible and ensure that as many tests are useful in the future as possible.

The actual process that the software team followed was crude due to having very little time. We made a function to essentially act as an example case that we manually checked. This is a useful function to have as it gives us a good amount of information, though formal tests would have been preferred.

5.4 Software Testing Results

The test was useful in our development process for giving us continual feedback whenever we needed to test if we broke anything. As our code is an ongoing project that will be continued past our group, we do not have any final formal results. At the time of creating our final code product for the class, our test function works properly.

5.5 Security Concerns

Hardware side: No immediate physical security concerns. Our only immediate concern from a design perspective is that our work is bespoke and cutting edge, so there are concerns regarding plagiarism / intellectual security.

If / when a physical computer based on this design is constructed, it will be afflicted with all regular hardware security concerns that befall a regular quantum computer. Intellectual security, tampering, natural disasters, and potentially a certain national security level concern are potential issues.

Software side: There are no software security concerns due to the nature of our code. The software should only be used by experts who know how to properly use it and the code isn't available to anyone who may misuse it. It is not uploaded to anywhere on the internet other than our private git.ece.iastate repo. The only potential concern could be with bugs in our code, though it would have little to no impact on anything security related.

6 Closing Material

This final section outlines our thoughts on the work we've done and how we see our proposed design evolving in the future. We also discuss how our design fits into the larger quantum computing ecosystem.

6.1 Conclusion

The results of our project are some beginning steps of a theorized quantum computer design, preliminary simulations provided by a graduate student who is attached to our project, and software to schedule operations on a quantum computer. These simulations produced promising results for our design as we were successful in our simulation of an ion handoff between two

radiofrequency (RF) segments. We have a good foundation on the software of scheduling operations, moving qubits, and controlling the quantum computer. We thoroughly fleshed out the measurements and actual "look" of a quantum computer of this specification, and overcame some large questions that weren't answered at the outset of this project. We have created some diagrams of our proposed innovation(s) relative to the industry's current leading Ytterbium ion trap quantum computer design and have applied the knowledge that we acquired last semester in CPRE 491. This semester we expected to expand what we had into a formal and thorough design with more robust simulations, which we did accomplish. We began work on an academic paper detailing our design. which should be finished in the near future. In due time, we hope a full scale, real-to-life simulation (and / or construction) of our design is created. If either is done, we plan on submitting a patent for our design.

6.2 Design Fitment in Quantum Computing Ecosystem

Advances in technology:

- Our design proposes to bring quantum computing to a new scale never before seen
    - The largest problem in quantum computing today is the inability to scale quantum computers, keeping computers reliable at high scale is difficult to achieve.
    - Quantum computers are not currently capable of realistically executing many of the identified quantum-superior algorithms due to their limited computational capacity.
    - Our structure of having nodes will allow for greater size to the order of a kilo-qubit scale
- We will implement memory qubits in our design
    - Qubits are only able to stably hold information for certain amounts of time, typically only a few cycles
    - Our design implements quasi-crystal memory qubits to vastly increase the cycles a qubit can stably hold information for
    - Memory qubits are ground breaking to advancing quantum computing because it allows for longer and more complex computations. Imagine if you could only hold a piece of memory in a classical computer for a few cycles. Extending the amount of cycles the information can be used for is vital to the success of a quantum computer

Relevant Quantum Computing Technologies:

- Similar Concepts: Our design is, to speak loosely, "spiritually" similar to a device known as a QCCD, which uses multiple connected segmented RF-electrode traps to shuttle a comparably smaller number of ions throughout their node-like structure.
- Implementations: The QCCD is a conceptual architecture, which has been most-successfully implemented by Honeywell's "Quantinuum" team in the form of their H1 and H2 machines. At time of writing, their architecture holds the world record for the highest computational capacity (Quantum Volume) of any quantum computer. This was one of the driving forces in pursuing our new paradigm.
- Differences: The fundamental difference of our client's proposed design is our ability to achieve "static" trap-to-trap ion handoff by having overlapping inverted traps, and the increase in the number of qubits-per-trap this design allows, due to the lessened need for

shuffling. These changes introduced the significant functional and operational concerns our client expected us to address.

6.3 References

A. Shaw, "Simulating Ion Trap Quantum Computers," *Researchgate.net*, Oct-2022. [Online]. Available: https://www.researchgate.net/publication/364316902_Simulating_Ion_Trap_Quantum_Computers. [Accessed: 01-Dec-2022].

D.-I. "D. Cho, S. Hong, M. Lee, and T. Kim, "A Review of Silicon Microfabricated Ion Traps for Quantum Information Processing," *Micro and Nano Systems Letters - A SpringerOpen Journal*, 2016.

D. J. Berkeland, J. D. Miller, J. C. Bergquist, W. M. Itano, and D. J. Wineland, "Minimization of Ion Micromotion in a Paul Trap," *Journal of Applied Physics*, vol. 83, no. 10, pp. 5025–5033, May 1998.

G. N. Nop, D. Paudyal, and J. D. H. Smith, "Ytterbium Ion Trap Quantum Computing: The Current State-of-the-Art," dissertation, 2021.

M. Heinrich, "On Stabiliser Techniques and their Application to Simulation and Certification of Quantum Devices," dissertation, University of Cologne , Cologne , North Rhine-Westphalia, 2021.

M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 2021.

P. Maunz, *High Optical Access Trap 2.0*. Sandia National Lab, 2016.

T. Jones, "Mathieu's Equations And The Ideal RF-Paul Trap." Accessed: Dec. 01, 2022. [Online]. Available: http://einstein.drexel.edu/~tim/open/mat/mat.pdf

W. C. Burton and B. Esty, "Transport of Multispecies Ion Crystals Through a Junction in an RF Paul Trap," *arXiv.org*, 24-Jun-2022. [Online]. Available: arXiv.org. [Accessed: 01-Dec-2022].

Z. An and D. L. Zhou, "Deep Reinforcement Learning for Quantum Gate Control," *EPL (Europhysics Letters)*, vol. 126, no. 6, p. 60002, 2019.

Z. D. Romaszko *et al.*, "Engineering of Microfabricated Ion Traps and Integration of Advanced on-Chip Features," *Nature Reviews Physics*, vol. 2, no. 6, pp. 285–299, May 2020, doi: 10.1038/s42254-020-0182-8.

7 Appendices

7.1 Operation Manual

The Digital Clone consists of three Python classes: Node, NodeLiteral, and Cycles. The Node class represents one Node in the quantum computer. There is no notion of a memory trap currently implemented. To build a Node with n qubits:

- node = Node(n)

The user can then specify what quantum algorithm they want the Node to run through a series of method calls to the Node. This builds a sequence of quantum operations for such a Node to carry out. Qubits are referenced by integers; only as many qubits as are specified in its construction may be used. The Node supports X, Y, Z, H, P(lambda), and CNOT gates, which should be sufficient to construct any quantum algorithm. To build such a quantum algorithm:

- node.sop(q, 'x') # An X gate on qubit q
- node.sop(q,'y') # A Y gate on qubit q
- node.sop(q,'z') # A Z gate on qubit q
- node.sop(q,'h') # A H gate on qubit q
- node.sop(q,'p',ph) # A phase gate on qubit q with a phase of ph
- node.mop(p,q) # A C-NOT gate between qubits p and q, where p is the controlling qubit

Once the user has constructed a quantum algorithm for a Node, the Node can generate a Node Literal for the quantum operation. The NodeLiteral consists of a collection of Cycles, which specify all quantum operations that should happen within each machine clock cycle. Our theoretical hardware design would be able to take advantage of this multitasking in a way that contemporary quantum computers cannot. The generated NodeLiteral also specifies qubit movement because multi-qubit gates must be adjacent to each other. To generate a NodeLiteral from a Node:

- literal = n.getLiteral()

In addition, descriptions of specific Nodes, NodeLiterals and Cycles may all be printed to stdout e.g. a command prompt or terminal. Descriptions of Nodes reveal the total number of qubits, and the series of quantum operations therein. To print the description of a Node:

- node.describe()

A NodeLiteral is also capable of running its quantum algorithm on contemporary quantum computers and simulations through qiskit. This does not take advantage of inter-cycle multitasking, which remains theoretical. Despite qubit movement, the qubits used in the execution should be roughly the same as those known to the Node class, erasing any gaps in qubits used, so if qubits 0, 1 and 3 are used, they should be assigned 0, 1 and 2. To execute a NodeLiteral's quantum algorithm on a backend:

- result = literal.execute(backend)

The user may then use or print the results from the backend as they would otherwise with qiskit. Both the backend object and result object come from qiskit and are not our own. It is advised not to alter this array. One way to print the results from the operations is:

- print(result.get_counts())

However, the user may wish to acquire the circuit running on the backend directly, for such purposes as observation. To acquire a circuit object directly without executing it on a backend, one may use:

- circ = literal.getcirc()

Descriptions of NodeLiterals reveal their total number of qubits, and descriptions of their cycles. To describe a NodeLiteral:

- literal.describe()

In the process of generating a node literal, the initial positions of the qubit will almost inevitably be different from their initial assignment known to the Node class. Knowledge of this ordering may be useful to understand the output of the describe() method. To get an array with the starting position of each logical qubit, one may use:

- order = literal.getOrder()

A user may also manually specify the composition of a NodeLiteral. This gives a greater deal of control to the user to specify how quantum operations would take place on a Node of our design. To construct a NodeLiteral:

- literal = NodeLiteral(n)

The user may wish to specify a certain qubit ordering at the beginning of the algorithm in order to end up with some other ordering by the end of the algorithm due to qubit movements. An initial qubit ordering is established by an array of integers referring to the designation of logical qubits within each location, so [1, 3, 0, 2] specifies that qubit 1 lies in position 0, qubit 3 lies in position 1, etc.. To set an initial qubit ordering in the circuit, use:

- literal.setOrder(order)

Unlike with the Node class, the user must create and add cycles to the NodeLiteral. These cycles would then be executed in series. To add a cycle:

- literal.addCycle(cycle)

A cycle specifies a sequence of commands that would be done in a single cycle in our hypothetical design. Within a cycle, each qubit may only be involved in at most one operation. To construct a cycle:

- cycle = Cycle(n)

A cycle may specify a number of single qubit operations, multi qubit operations, and qubit movement operations. Both multi qubit operations and qubit movement operations can only be

performed between adjacent qubits e.g. between qubits 2 and 3. Any qubit may not be used more than once within a cycle. To add these to a cycle:

- cycle.sop(q,'x') # Applies an X gate to qubit q
- cycle.sop(q,'y') # Applies a Y gate to qubit q
- cycle.sop(q,'z') # Applies a Z gate to qubit q
- cycle.sop(q,'h') # Applies a H gate to qubit q
- cycle.sop(q,'p',ph) # Applies a phase gate to qubit q with a phase of ph
- cycle.mop(p,q) # Applies a C-NOT gate between qubits p and q, with qubit p as the control.
- cycle.mvq(p,q) # Swaps qubits p and q

Descriptions of Cycles reveal their size, single qubit operations, multi qubit operations and qubit movement operations. To describe a Cycle:

- cycle.describe()

7.2 Code

Node.py

```
import qiskit
from qiskit import QuantumCircuit
from qiskit import BasicAer
from Cycle import Cycle
from NodeLiteral import NodeLiteral

class Node:
    """
    Given a sequence of instructions, computes a dependency
    mapping between operations (temporal) and resources (physical)
    and constructs cycles for use in the node literal.

    Concept:
    Initially, operations can be conducted on any qubit/qubit pair in
    the machine. As execution progresses, the need to reference these
    qubits that have had specific operations done on them again will
    impose constraints/require qubits to be moved.
    Although target bits for an initial operation can be arbitrary,
    some choices for initial target bits will more easily comply
    with future needs, and will result in faster execution/less noise
    and error from qubit movement. Our goal is to identify these preferable
    operation and resource mappings.

    GRAPE algo: overkill, or necessary?
    (We did not intentionally use a GRAPE algorithm)

    Concept 2: a simplified view of the decoherence problem:
        Qubit information decays over time, and moreso when they move.
        Given a sequence of operations, we can achieve optimal gate
        sequences using the GRAPE algo to limit movement and general decay.

        However, if we have a medium-scale algo there's a chance that
        we will need a mqop between using that can't be brought together before
        one/both decay. We need to recognize this, and account for it by
        implementing ops that get q1 into memory while q2 is brought/prepared
```

```
nearby.
            -This raises another issue: can q1 be retrieved from memory before q2
decays?
                -q1 can only be retrieved on fib. cycles. How can q2 be made ready
(max coherence)
                right when q1 is retrieved?

    Questions: what will "setting up" the q2 look like? How to know when it's
necessary?
    """

    fib = [8, 13, 21, 34]
    def __init__(self,size:int):
        self.size = size                        #
        self.using = [False]*self.size          #Array of which qubits are being
used
        self.nused = 0                          #Number of used qubits
        self.ops = []                           #Array of operations
        return

    def sop(self,p,op,ph=0):        #Single qubit operations
        if not self.using[p]:       #If qubit isn't being used, then it should be
            self.using[p] = True    #Now using the qubit
            self.nused += 1         #Number of qubits being used
        self.ops.append([op,p,ph])  #Add the operation, even if the qubit is already
used
        return self

    def mop(self,p,q):              #Multiple qubit operations
        if not self.using[p]:       #If qubit isn't being used, then it should be
            self.using[p] = True    #Now using the qubit
            self.nused += 1         #Update # of used qubits
        if not self.using[q]:
            self.using[q] = True
            self.nused += 1
        self.ops.append(["cx",p,q]) #Add the operation. cx is the only multiple
qubit operation so it is hard coded
        return self

    def describe(self):
        print("Size:   "+str(self.size))
        print("Ops: "+str(self.ops))
        return self

    def getLiteral(self):
        rf = [0] * self.size # Most Recently Free
        lq = [] # Location of Qubit on a given cycle (2D)
        ql = [] # Inverse of lq
        lf = [] # Location Freed Status on a given cycle (2D) (0->free nonzero->op
id)
        id = 1
        cycles = []
        lq.append([-1] * self.size)
        ql.append([-1] * self.size)
        lf.append([0] * self.nused)
        cycles.append(Cycle(self.size))
        j = 0
        for i in range(self.size):      #Of all physical qubits
            if self.using[i]:           #If they are being used
                lq[0][i] = j            #Provide an initial mapping of the location
of qubit being used
```

```
                ql[0][j] = i
                j += 1
        n = NodeLiteral(self.size)
        for i in range(len(self.ops)-1,-1,-1):
            if self.ops[i][0] == "cx":
                x = self.ops[i][1]
                y = self.ops[i][2]
                r = max(rf[x],rf[y])-1
                if len(lq) == r:    # If x/y not free at last cycle, add cycle
                    lq.append(lq[r-1].copy())
                    ql.append(ql[r-1].copy())
                    lf.append([0]*self.nused)
                    cycles.append(Cycle(self.size))
                p = lq[r][x]
                q = lq[r][y]
                if p < q: inc = 1
                else: inc = -1
                while abs(p-q) != 1:
                    f = len(lq)
                    if f <= r+1:
                        lq.append(lq[r-1].copy())
                        ql.append(ql[r-1].copy())
                        lf.append([0]*self.nused)
                        cycles.append(Cycle(self.size))
                        f += 1
                    if lf[r][p]==0 and lf[r][p+inc]==0:
                        cycles[r].mvq(p,p+inc)
                        lf[r][p] = id
                        lf[r][p+inc] = id
                        for j in range(r+1,f):
                            z = ql[r][p+inc]
                            lq[j][x] = p+inc
                            lq[j][z] = p
                            ql[j][p] = z
                            ql[j][p+inc] = x
                        id += 1
                        p += inc
                    elif lf[r][p] == lf[r][p+inc]:
                        p += inc
                    if abs(p-q) == 1: break
                    if lf[r][q]==0 and lf[r][q-inc]==0:
                        cycles[r].mvq(q,q-inc)
                        lf[r][q] = id
                        lf[r][q-inc] = id
                        for j in range(r+1,f):
                            z = ql[r][q-inc]
                            lq[j][y] = q-inc
                            lq[j][z] = q
                            ql[j][q] = z
                            ql[j][q-inc] = y
                        id += 1
                        q -= inc
                    elif lf[r][q] == lf[r][q-inc]:
                        q += inc
                    r += 1
                while lf[r][p]!=0 or lf[r][q]!=0:
                    r += 1
                    f = len(lq)
                    if f <= r:
                        lq.append(lq[r-1].copy())
                        ql.append(ql[r-1].copy())
```

```
                        lf.append([0]*self.nused)
                        cycles.append(Cycle(self.size))
                        f += 1
                rf[x] = r + 1
                rf[y] = r + 1
                lf[r][p] = id
                lf[r][q] = id
                id += 1
                cycles[r].mop(p,q)
            else:
                x = self.ops[i][1]
                op = self.ops[i][0]
                ph = self.ops[i][2]
                r = rf[x]-1
                p = lq[r][x]        # Add x's new op and update node MData.
                while lf[r][p]!=0:
                    r += 1
                    f = len(lq)
                    if f <= r:    # If x not free at last cycle, add cycle
                        lq.append(lq[r-1].copy())
                        ql.append(ql[r-1].copy())
                        lf.append([0]*self.nused)
                        cycles.append(Cycle(self.size))
                    p = lq[r][x]        # Add x's new op and update node MData.
                rf[x] = r + 1
                lf[r][p] = id
                id += 1
                cycles[r].sop(p,op,ph)
        for i in range(len(cycles)-1,-1,-1):
            n.addCycle(cycles[i])
        n.setOrder(lq[len(lq)-1])
        return n

"""
n = Node(8)
n.sop(0,'x')
n.sop(1,'y')
n.sop(0,'z')
n.sop(2,'x')
n.sop(2,'z')
n.mop(0,3)
n.sop(0,'p',.5)
l = n.getLiteral()
print("DESCRIPTION")
l.describe()

circ = l.getcirc()
print(circ.draw('text'))

backend = BasicAer.get_backend('qasm_simulator')
r = l.execute(backend)
print(r.get_counts())
print("END PROGRAM")
"""
```

NodeLiteral.py

```
import qiskit
from qiskit import QuantumCircuit
from qiskit import BasicAer
```

```
import Cycle
from Cycle import Cycle

class NodeLiteral:
    """
    NodeLiteral: The 'hardware' level node. Direct interaction with qubits,
    no logic, controls, optimizations, etc.

    @param size: no. of qbits in the node

    """

    fib = [8, 13, 21, 34]
    def __init__(self,size:int):
        self.size = size
        self.cycles = []
        self.order = [0] * self.size
        for i in range(self.size):
            self.order[i] = i
        return

    def addCycle(self,cycle):
        if self.size == cycle.size:
            self.cycles.append(cycle)
            return self
        else:
            return None

    def describe(self):
        print("Cycles: "+str(len(self.cycles)))
        print("Order: "+str(self.order))
        for c in self.cycles:
            c.describe()

    def setOrder(self,order):
        self.order = order

    def getOrder(self):
        return self.order

    def getcirc(self):
        circ = QuantumCircuit(self.size)
        qb = list(range(self.size))
        qb = self.order.copy()
        cnt = 0
        for cycle in self.cycles:
            for op in cycle.sqops:
                """
                if (op[0]%(self.tsize+1))==self.tsize:
                    print("ERROR: SQ: Memory Node Selected")
                    return None
                """
                if op[1] == "x":
                    circ.x(qb[op[0]])
                elif op[1] == "y":
                    circ.y(qb[op[0]])
                elif op[1] == "z":
                    circ.z(qb[op[0]])
                elif op[1] == "h":
                    circ.h(qb[op[0]])
                elif op[1] == "p":
```

```
                circ.p(op[2],qb[op[0]])
            for op in cycle.mqops:
                """
                for p in op:
                    if (p%(self.tsize+1))==self.tsize:
                        print("ERROR: MQ: Memory Node Selected")
                        return None
                """
                if abs(op[0]-op[1]) == 1 or abs(op[0]-op[1]) == size.self-1:
                    circ.cx(qb[op[0]],qb[op[1]])
                else:
                    print("ERROR: CN: Qubits not Adjacent")
                    return None
            for op in cycle.mvqbt:
                """
                for p in op:
                    if (p%(self.tsize+1))==self.tsize:
                        c = self.mem[(p-self.tsize)/self.traps]
                        if c == -1 or cnt - c in self.fib:
                            self.mem[(p-self.tsize)/self.traps] = cnt
                        else:
                            print("ERROR: MV: Memory Node Not on Fib Cycle")
                            return None
                """
                if abs((op[0]%self.size)-(op[1]%self.size)) == 1:
                    tmp = qb[op[0]]
                    qb[op[0]] = qb[op[1]]
                    qb[op[1]] = tmp
                else:
                    print("ERROR: MV: Qubits not Adjacent")
                    return None
            cnt += 1
        circ.measure_all()
        return circ

    # Execute jobs specified for this node.
    def execute(self,backend):
        circ = self.getcirc()
        oc = qiskit.transpile(circ, backend)
        job = backend.run(oc)
        return job.result()


"""
n = NodeLiteral(8)
c = Cycle(9)
c.sop(0,"x")
c.mop(1,2)
c.mvq(3,4)
n.addCycle(c)
n.describe()

backend = BasicAer.get_backend('qasm_simulator')
r = n.execute(backend)
print(r.get_counts())

print("END PROGRAM")
"""
```

Cycle.py

```
# Important: operations that happen within each cycle cannot
#     share a single qubit
from __future__ import annotations
class Cycle:
    """
    A class to simulate a cycle of the quantum computer.
    We treat cycle as a temporal snapshot, where all operations
    are executed more-or-less at the same time.

    If an operation is added, the involved qubits are marked
    as un-available for any other operations during the cycle.

    @ param n: Number of qubits

    """
    def __init__(self,size):
        self.size = size
        self.qbits = [True] * size
        self.sqops = []
        self.mqops = []
        self.mvqbt = []
        return

    def sop(self,p,op,ph=0):
        if (self.qbits[p]):
            self.qbits[p] = False
            self.sqops.append([p,op,ph])
            return self
        else:
            print("SOP ERROR: Qubit Reserved")
            return None

    # TODO: qbits need to be next to each other and available
    def mop(self,p:int,q:int):
        if (self.qbits[p] and self.qbits[q]):
            self.qbits[p] = False
            self.qbits[q] = False
            self.mqops.append([p,q])
            return self
        else:
            print("MOP ERROR: Qubit Reserved")
            return None

    def mvq(self,p,q):
        if (self.qbits[p] and self.qbits[q]):
            self.qbits[p] = False
            self.qbits[q] = False
            self.mvqbt.append([p,q])
            return self
        else:
            print("MVQ ERROR: Qubit Reserved")
            return None

    def describe(self):
        print("Size:   "+str(self.size))
        print("Qubits: "+str(self.qbits))
        print("SQ Ops: "+str(self.sqops))
        print("MQ Ops: "+str(self.mqops))
        print("Qbt Mv: "+str(self.mvqbt))
        return self
```