Quantum Computation: Theory and Implementation

by

Edward Stuart Boyden III

Submitted to the Department of Physics
in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Physics
and to
the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science and
Master of Engineering in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology

May 7, 1999

Author _____

Edward S. Boyden III
Physics and Media Group, MIT Media Laboratory
Department of Physics
Department of Electrical Engineering and Computer Science

Certified by _____

Neil Gershenfeld
Professor of Media Technology, MIT Media Laboratory
Thesis Supervisor

Accepted by _____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Accepted by _____

Prof. David E. Pritchard
Senior Thesis Coordinator, Department of Physics

Table of Contents

0.1 Abstract

Quantum Computation: Theory and Implementation

by

Edward Stuart Boyden III

Submitted to the Department of Physics
in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Physics
and to
the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science and
Master of Engineering in Electrical Engineering and Computer Science
May 7, 1999

Quantum computation is a new field bridging many disciplines, including theoretical physics, functional analysis and group theory, electrical engineering, algorithmic computer science, and quantum chemistry. The goal of this thesis is to explore the fundamental knowledge base of quantum computation, to demonstrate how to extract the necessary design criteria for a functioning tabletop quantum computer, and to report on the construction of a portable NMR spectrometer which can characterize materials and hopefully someday perform quantum computation. This thesis concentrates on the digital and system-level issues. Preliminary data and relevant code, schematics, and technology are discussed.

Supervisor: Neil Gershenfeld
Title: Professor of Media Technology

0.2 Acknowledgements

This project would have been impossible without the help and guidance of Neil Gershenfeld, Matt Reynolds, Rich Fletcher, Ben Vigoda, Rehmi Post, Bernd Schoner, Joey Richards, Chris Douglas, and the rest of the amazing PHM group, which is certainly amongst the finest set of hackers and engineers on the planet. Yael Maguire was responsible for most of the insight on the magnetics, and he also designed the Mark I Quantum Computer analog signal chain last summer. Susan Bottari also deserves credit for being an excellent administrator and facilitator. Those PHMers who came before, like Josh Smith, Henry Chong, Chris Verplaetse, and Matthew Gray, indirectly helped form the direction this research took. Neil's collaborators, sponsors, and friends, especially Ike Chuang, Glenn Vonk, and Simon Capelin, helped motivate certain directions that this thesis has taken.

May the future of quantum computing be as promising as its short and exciting past.

"Because it's there!" – *George Mallory, before dying on Mt. Everest, on why he wanted to climb it*

I. Introduction

Why build a quantum computer? Because it's *not* there, for one thing, and the theory of quantum computation, which far outstrips the degree of implementation as of today, suggests that a quantum computer would be a incredible machine to have around. It could factor numbers exponentially faster than any known algorithm, and it could extract information from unordered databases in square-root the number of instructions required of a classical computer. As computation is a nebulous field, the power of quantum computing to solve problems of traditional complexity theory is essentially unknown. A quantum computer would also have profound applications for pure physics. By their very nature, quantum computers would take exponentially less space and time than classical computers to simulate real quantum systems, and proposals have been made for efficiently simulating many-body systems using a quantum computer. A more exotic application is the creation of new states which do not appear in nature: for example, a highly entangled state experimentally unrealizable through other means, the 3-qubit Greenberger-Horne-Zeilinger state

$$|000\rangle + |111\rangle,$$

has been prepared on an NMR quantum computer with only 3 operations, a $Y(\pi/2)$ pulse followed by 2 CNOTs, applied to the spins of trichloroethylene (Laf97). Computational approaches to NMR are interesting for their own sake: Warren has coupled spins as far as millimeters apart, taking advantage of the properties of a phenomenon known as zero-quantum coherence to make very detailed MRI images (War98). Performing fixed-point analyses on iterated schemes for population inversion can result in very broadband spin-flipping operators – in such a scheme, the Liouville space of operators is contracted towards a fixed point, so that operators that are imperfect (due to irregularities in the magnetic field, end effects of the stimulating coil, impurities in the sample, etc.) nevertheless result in near-perfect operations (Tyc85) (Fre79). These simple uses of nonlinear dynamics and the properties of solution have shed light on certain computational aspects of nuclear magnetic resonance.

But performing a quantum computation as simple as factoring 15 will involve the creation and processing of dozens of coherences over extended periods of time. This presents an incredible engineering challenge, and may open up new understandings on both computation and quantum mechanics.

Undoubtedly we will someday have quantum computers, if only to maintain progress in computation. It has been widely speculated that within a few decades, the incredible growth which characterizes today's silicon age will come to a gradual halt, simply because there will be no more room left at the bottom: memory bits and circuit traces will essentially be single atoms on chips, the cost of a fabrication plant will be "the GNP of the planet," and there will probably be lots of interesting problems to solve which require still more computing power. This power will have to come from elsewhere – that is, by stepping outside of the classical computation paradigm.

Other fields related to quantum computing, which might be said to fall into the newborn field of ‘quantum control,’ include quantum teleportation, quantum manipulation, and quantum cryptography. Quantum teleportation is an appealing way to send quantum states over long distances, something no amount of classical information can ever do (probably), and has been realized for distances exceeding 100 kilometers (Bra96). Quantum means of cryptography are unbreakable according to the laws of physics, and using quantum states to send messages guarantees that eavesdroppers can be detected (Lo99).

Furthermore quantum computing joins two of the most abstruse and surprisingly nonintuitive subjects known to humans: computation and quantum mechanics. Both have the power to shock the human mind, the former with its strange powers and weaknesses, and the latter with its nonintuitive picture of nature. Perhaps the conjunction of these two areas will yield new insights into each. For the novice in this field, past overview papers on quantum computing include (Bar96) (Llo95) (Ste97).

There are powerful alternatives to quantum computing which may have more immediate significance for the computing community. Incremental improvements in silicon fabrication (such as MEMS-derived insights, low-power and reversible design principles, copper wiring strategies, intergate trench isolation, and the use of novel dielectrics), reconfigurable computing and field programmable gate arrays, single-electron transistors, superconducting rapid single-flux quantum logic, stochastic and defect-tolerant computers, ballistically-transporting conductors, printable and wearable computers, dynamical systems capable of computation, memory devices consisting of quantum dots and wells, and many other technologies are vying for the forefront. Any of them, if lucky and well-engineered, could jump to the forefront and perhaps render the remainder obsolete, much like the alternatives to silicon in the heyday of Bardeen and Intel. It is more likely that these diverse technologies will find a variety of niches in the world, given the amazing number of problems that people want to solve nowadays.

This paper begins with a comprehensive critique of much of the knowledge in the quantum computation community, since that is perhaps fundamental to understanding why and how quantum computers should be built. Part **I.1.** covers elementary quantum mechanics, information theory, and computation theory, which is a prerequisite for what follows. Much of this was written over the past year to teach the author how all of these pictures can contribute to a coherent picture of the physics of computation. Part **I.1.** can also be thought of an ensemble of collected material, in preparation for a book which delves into the theories and experiments behind the physics of computation; it is the opinion of the author that the definitive book on the physics of computation has not yet been written, and that the world needs one. It is noted that very little synthesis has been performed on the information here presented; most of the paragraphs are summaries of various different sources, assembled for easy reference and as an aid to thinking. Some of these notes were presented at (Bab98).

I.1. Background

I.1.i. Quantum mechanics

The axioms of nonrelativistic quantum mechanics are simple, fewer in number than those of

geometry, but they demand great credulity on the part of the aspiring physicist. It is possible to cast the axioms in an extremely abstruse form: for example, (Per) starts by stating ten axioms, beginning with:

> 1) If a system yields a definite outcome in a maximal measurement (one whose result accounts for as much as can possibly be tested at an instant), then all other measurements have values that occur with definite probabilities.
> 2) Systems with N states can be prepared such that each outcome of a maximal test occurs with probability 1/N. We have the least prior knowledge about this state.
> 3) If *a*, *b* are pure states as measured with operators A, B respectively, then the outcome of measuring *b* with operator A has the same probability distribution as that of measuring *a* with operator B.
> 4) If a system can follow several paths from preparation to measurement, then the probability of the outcome is not the sum of the probabilities over paths…

and so on. It's more concise just to say that quantum mechanics behaves like a linear vector space with complex amplitudes, the squared magnitudes of which correspond to probabilities, and the time-dynamics of which are determined by the Schrodinger equation and the fundamental symmetries and statistics of the particles involved. A more precise formulation of this statement, as well as experimental and logical justification for these assertions, can be found in (Sak) or in (Lib). A nice discussion of probabilities, states, measurements, and distinguishability is given in (Fuc96), which also contains 528 references for the aspiring quantum-information theoretician.

Interestingly, many people have tried to come up with smaller sets of axioms, trying to avoid artificial-looking requirements like superselection rules (in which, for example, superpositions of states with different charges or different statistics are forbidden), the phenomenon of wavefunction collapse (where ultimately a human measurement causes, by some miracle of consciousness, a superposition of states to assume a deterministic configuration), and the contrived nature of observables (which require non-dynamical, instantaneous, pseudoclassical properties, as opposed to a simply flowing dynamical picture of the universe). The "program of decoherence" tries to explain all of these things by interactions with the environment: superselection rules are explained by implicit measurements (e.g., a superposition of different charge states would interact with, and therefore be 'measured' by, all the other charges in the universe, according to Maxwell's equations), wavefunction collapse is explained by the entanglement of a system under study with a measuring device of great complexity, and observables can be constructed directly from quantum states, rather than relying on classical probabilities and expectation values as the fundamental concepts (Giu). This is a delightful area of nonintuitive and speculative physics, with much room for philosophical and mathematical lines of logical thought.

We will not consider the remarkable constraints on the behavior of nature that are incurred by invoking the principle of Lorentz-invariance (Pes). Consistently dealing with special relativity leads to phenomena such as antiparticles, tachyons, the spin-statistics theorem, quantum electrodynamics and chromodynamics, and subtle issues concerning renormalization of infinities and the meaning of observables like the mass of a particle. One can only speculate what a

competent theory of quantum mechanics in curved spacetime might look like! See (Bae) for one prominent quantum gravity theoretician's diary of his explorations in this field.

Throughout this document we will denote pure states by $|x\rangle$, where *x* indicates the eigenvalue corresponding to a certain quantum mechanical state, defined with respect to the eigenvector basis of a Hermitian measurement operator. We define the qubit basis to be $|1\rangle$, $|0\rangle$, which can correspond to the up and down states of a spin or an electric dipole, or to any two-state system, such as an atom being irradiated with radiation at a frequency such that only one transition is stimulated. The Bell basis is defined to be

$$|+\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle),\ |-\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle),$$

and can be thought of as the state of a qubit that has been rotated by 45 degrees. We assume familiarity with the basic facts of algebra, such as the Spectral Theorem for unitary operators, which says that one can decompose a unitary *U* into P^*DP where *D* is diagonal and *P* unitary. The elementary vector (0, 0, …, 1, …, 0, 0, 0) is represented by e_a when the 1 is in the a^{th} slot. The reader in need of a mathematical review is well advised to spend some time studying (Art) or (Str).

We assume some familiarity with the spinor, the 2-dimensional vector (*a*, *b*) that represents the quantum state $a|0\rangle + b|1\rangle$, where *a* and *b* are complex numbers whose squared magnitudes sum to 1. Throughout the text of this document we will use the Dirac matrices, $\vec{\sigma} = \{I, \sigma_1, \sigma_2, \sigma_3\}$, defined so that the spin operators are given by $I_x = \hbar/2\sigma_1$, $I_y = \hbar/2\sigma_2$, and $I_z = \hbar/2\sigma_3$, such that

$$\sigma_1 = \begin{bmatrix} & 1 \\ 1 & \end{bmatrix},\ \sigma_2 = \begin{bmatrix} & -i \\ i & \end{bmatrix},\ \sigma_3 = \begin{bmatrix} 1 & \\ & -1 \end{bmatrix}$$

(By convention, nondeclared matrix entries are assumed to be zero.) These matrices are sometimes labeled σ_x, σ_y, and σ_z, respectively. They are the generators of the Lie algebra of the 3D rotation group, in its spinor representation (Sat). We derive these rotations in **II.2.**, for the benefit of the reader. For NMR quantum computing simulation purposes, Matlab code for the rotation operators is provided in **Appendix B**, along with some supporting and example code. Armed with these definitions, we define the raising and lowering operators I^+ and I^-,

$$I^+ = I_x + iI_y = \hbar \begin{bmatrix} & 1 \\ & \end{bmatrix},\ I^- = I_x - iI_y = \hbar \begin{bmatrix} & \\ 1 & \end{bmatrix}$$

Note that $[\sigma_i, \sigma_j] = i\hbar\,\sigma_k$, where [,] denotes the commutator and *ijk* is an even permutation of *123* (Lib). Also $\sigma_i^2 = 1$, and

$$I^+I^- = \hbar^2 \begin{bmatrix} 1 & \\ & \end{bmatrix},$$

the 'number operator' for a fermionic system. The identity matrix will usually be referred to as 1 or I. The conjugate transpose of a unitary matrix *U* will be denoted by $U^* = U^{-1}$. For an *n*-dimensional Hilbert space, the space of operators on that Hilbert space is a n^2-dimensional space often called the Liouville space. (To see that it is indeed n^2-dimensional, simply note that an operator on an *n*-dimensional Hilbert space can always be written as an $n \times n$ matrix, which has n^2 entries.)

Inspired by the analogous Matlab command, we will often refer to the matrix

$$\begin{bmatrix} a & & & \\ & b & & \\ & & \ldots & \\ & & & c \end{bmatrix}$$

as diag([*a*, *b*, …, *c*]) for shorthand. Also in accordance with Matlab notation, we will occasionally refer to the matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

as [*a*, *b*; *c*, *d*] or even [*a b*; *c d*]. Since we use density matrix models throughout this paper, a working familiarity with Matlab may not only clarify many of the examples given, it may help the reader discover new angles on what is presented here. We take the convention that unitary matrices operate upon a state vector on the left, so that if a product of matrices operates upon a state vector, the rightmost operator occurs earliest in time. Skewed indices on a product indicate the order in which matrices are to be written:

$$\prod_{j=0}^{3} U_j = U_0 U_1 U_2 U_3,$$

so that in this case, as the index variable increases, the matrices appear from left to right. Note that in this example, the matrix U_3 operates on the system earliest in time.

The *density matrix* ρ is a crucial element in the picture of quantum mechanics that we will adopt. For a pure state

$$|\psi\rangle = \sum_i c_i |i\rangle,$$

define the matrix element ρ_{ij} by $\langle\psi|j\rangle\langle i|\psi\rangle = c_i c_j^*$, and the density operator by the *projection operator*

$$\rho = \sum_{i,j} c_i c_j^* |i\rangle\langle j|.$$

Then for any observable *A*, with projection operator representation

$$A = \sum_{i,j} A_{ij} |i\rangle\langle j|,$$

we have an expression for the expectation value of *A*, $\langle A\rangle = \mathrm{Tr}(A\rho)$, as can be seen by direct evaluation with the projection operator forms of ρ and *A*. Clearly $\mathrm{Tr}(\rho) = 1$, since $\rho_{ii} = |c_i|^2$. For a pure state, $\rho = \rho^2$, or equivalently, ρ has all zero eigenvalues except for one 1 (since for any pure state, we can transform to a basis where the state in question is one of the basis states, and the others can be found via Gram-Schmidt orthogonalization). The time evolution of a density matrix, for a pure state given by the expression for ψ above, is given by the time-dependent Schrodinger equation; the time-evolution equations for the coefficients are

$$i\hbar \frac{\partial c_i}{\partial t} = \sum_j H_{ij} c_j \rightarrow i\hbar \frac{\partial \rho_{ij}}{\partial t} = \sum_k H_{ik}\rho_{kj} - \rho_{ik} H_{kj} \rightarrow i\hbar \frac{\partial \rho}{\partial t} = [H, \rho]$$

where the last expression is just the negative of a typical ‘Heisenberg picture’ time evolution equation for an operator (the sign difference results from the fact that the operator ρ is not an observable, although it is Hermitian). If we are given a time-evolution operator

$$U(t_1, t_0) = e^{iH(t_1 - t_0)/\hbar}$$

where H is the Hamiltonian of the quantum system during the period from time t_0 to time t_1, then the density matrix evolves as

$$\rho(t_t) = U\rho(t_0)U^{-1}.$$

This equation is in many ways fundamental to our picture of NMR, since rotations of spins about spatial axes, interactions via J-coupling, and free precession, the three main processes that happen to spins, all have very convenient time-evolution operator representations. The resultant picture, the *product operator picture*, is described in **IV.1.ii.**

If we have a *mixed state* then we can construct a density matrix as follows: suppose that we have an ensemble of systems such that a single system is in state $|i\rangle$ with classical probability w_i,

$$\sum_i w_i = 1.$$

The states $|i\rangle$ need not be orthonormal. Then in any basis $|a\rangle$, the density matrix element ρ_{ab} is defined to be

$$\rho_{ab} = \sum_i w_i \langle a|i\rangle\langle i|b\rangle = \overline{c_a c_b^*};$$

where the bar denotes averaging over the ensemble. By inspection it is clear that $\langle A\rangle$ still is given by $\mathrm{Tr}(\rho A)$. Note that different mixtures of pure states which have the same density matrix are completely indistinguishable, since any observable's expected value can be expressed as a functional of the density matrix. If the bases $|a\rangle$ and $|i\rangle$ are the same orthonormal basis, then

$$\rho_{ij} = w_i \delta_{ij};$$

the fact that the off-diagonal terms are zero is called the *hypothesis of random phases*, and is fundamental to quantum statistical mechanics. It should be clear that ensemble time evolution is formally identical to pure state time-evolution, as long as components of the ensemble don't interact with one another.

As we will see, density matrices offer a good notational system for quantum channels, parts of entangled states, and bulk quantum computation. For example, consider a pure state ψ of two entangled particles, whose orthonormal basis states we will denote as $|i\rangle$, $|j\rangle$, $|k\rangle$, . . . and $|a\rangle$, $|b\rangle$, $|c\rangle$, . . ., so that

$$|\psi\rangle = \sum_{i,a} c_{ia}|ia\rangle,$$

which we can write in density matrix form as

$$\rho = \sum_{a,b,i,j} c_{ia} c_{jb}^* |ia\rangle\langle jb|.$$

If we have only access to the particle bearing the $|i\rangle$ states, then we must take the *partial trace* of the density matrix ρ, since we can never directly observe any property of the $|a\rangle$ states. The result is that the system we see is described by the density matrix

$$\rho_A = \mathrm{Tr}_A(\rho) = \sum_{a,b} \left(\sum_k c_{ak} c_{bk}^*\right) |a\rangle\langle b|,$$

since

$$(\rho_A)_{rs} = \sum_i \langle ri|\rho|si\rangle = \sum_{k,a,b,j,i} c_{ia} c_{jb}^* \langle rk\|ia\rangle\langle jb\|sk\rangle = \sum_{k,a,b,j,i} c_{ia} c_{jb}^* \delta_{ri}\delta_{ka}\delta_{js}\delta_{kb} = \sum_k c_{rk} c_{sk}^* .$$

Therefore for a pure wavefunction of two particles that cannot be expressed as a direct product, the individual-particle distributions can look like mixed states! This is essential for a proper understanding of quantum information theory, and to an understanding of quantum measurement. In fact, there is an area of quantum information theory concerned with *purifications* – that is, given a mixed state ρ_0, finding a pure state ψ_0 containing a subsystem B such that

$$\mathrm{Tr}_B |\psi_0\rangle\langle\psi_0| = \rho_0 .$$

One nice theorem concerning any mixed states ρ_0, ρ_1 with respective purifications ψ_0, ψ_1 is that

$$|\langle\psi_0|\psi_1\rangle| \le \mathrm{Tr}\sqrt{\rho_1^{1/2}\rho_0\rho_1^{1/2}}$$

(Uhl, quoted in Fuc96), which resembles several results on *quantum fidelities* that we will discuss later, in **I.2.ii.**

For example, the spinor (a, b) can be encoded in 3 entangled qubits as

$$a(|000\rangle + |011\rangle + |101\rangle + |110\rangle) + b(|111\rangle + |100\rangle + |010\rangle + |001\rangle).$$

Since the density operator of any one of the 3 qubits (found by taking the partial trace of the entangled state over the other two) is $|0\rangle\langle 0| + |1\rangle\langle 1|$, one cannot find out any information about a or b by measuring any one qubit! (However, one can rewrite $|000\rangle + |011\rangle + |101\rangle + |110\rangle \pm (|111\rangle + |100\rangle + |010\rangle + |001\rangle)$ as a direct product of 3 Bell states, so this does not suffice for general quantum error-correction; see section **I.1.ii.** for more information on this.)

Finally, maximizing the von Neumann entropy $-\mathrm{Tr}(\rho \ln(\rho))$ while holding the energy (i.e. the expected value of the Hamiltonian) constant, and constraining the trace of ρ to be 1 with a Lagrange multiplier, we get

$$\rho = e^{-\beta H}/Z,$$

the canonical ensemble, which is usually found by constraining the temperature to be constant and expanding the probability of a state in a Taylor series. To see that the von Neumann entropy is the correct measure of disorder of a system, we argue after the master himself (Neu55) as follows.

Consider an ensemble of separate identical systems S_i with respective states s_i, all at rest in a closed system, with the ability to exchange energy with each other: this ensemble is formally equivalent to an ideal gas. We will use such notions as 'volume' and 'number of particles' to make this equivalence more concrete. Let the density matrix describing the classical distribution of the systems s_i, as well as the quantum amplitudes within each s_i, be ρ. We will design a reversible transformation that converts this system ρ into another state ρ', and this will then provide us with a measure of the entropy difference between these two systems.

First, note that all pure states (with density matrices equal to projection operators P_s, e.g. $P_s = P_s^2$) have the same entropy. To see this, consider the following reversible method for converting $\rho = P_a$ into $\rho' = P_b$: assume states a, b are orthogonal (if not, we can subtract out the common,

parallel part and leave it alone, manipulating only the orthogonal componenets), and let the states b_k, $k = 1 \ldots n$, be equal to

$$b_k = \cos\left(\frac{\pi k}{2n}\right)a + \sin\left(\frac{\pi k}{2n}\right)b$$

where n is some integer. Extend each state b_k to an orthonormal set $\{ b_{k,i} \}$, and let R_k be an operator whose set of eigenvectors is $\{ b_{k,i} \}$. Also note that $\langle b_k | b_{k+1} \rangle = \cos(\pi/2n)$. Then measuring P_a with R_1, R_2, … R_n in successive measurements, results in P_b with amplitude arbitrarily close to 1, since

$$\lim_{n\to\infty} \cos(\pi/2n)^{2n} = 1 .$$

Since the process is reversible (we can always perform the measurements in reverse order), all pure states have the same entropy, say 0, in the thermodynamic limit.

Let us derive the equation $H = -\mathrm{Tr}(\rho \ln(\rho))$, where H is the von Neumann entropy of a quantum system. Consider an ideal gas of N molecules in volume V with density matrix

$$\rho = \sum_n w_n P_n ,$$

where $\{ P_i \}$ is a basis of eigenvectors in the projection operator representation. Separate the different components s_i into different boxes S_i (using semipermeable membranes that only let one state of the spectrum through, for example by adjoining an empty container and moving semipermeable walls through the gas so as to filter just one component at a time into the part of the empty container terminated by the appropriate walls). The result is a system of separated gases P_1, P_2, … with $w_1 N$, $w_2 N$, … molecules respectively. Now isothermally compress each component i to volume $w_i V$, thus increasing the entropy of the environment by

$$Nk \sum_n w_n \ln(w_n) .$$

Finally, one can, by the measurement method describe above, reversibly transform all the P_i components into P_0 – which, as we noted, has *zero entropy*. It follows that the original entropy must have been

$$H = -Nk \sum_n w_n \ln(w_n)$$

which is precisely $-Nk\ \mathrm{Tr}(\rho \ln(\rho))$ in a basis where ρ is diagonal. It is interesting to note that in the expression for H, the internal w_n (or eigenstate of ρ) comes from the isothermal compression step, whereas the external w_n (or eigenstate of ρ) comes from the percentage of the original composite system which is due to each individual component.

Interestingly, this shows that discarding a component of an entangled pure state is an irreversible process, since the entropy of the environment goes up in this process. Measurement is similarly a potentially irreversible process.

This has interesting consequences - for example, a membrane permeable to particles in state a but not particles in state b, can only exist if a and b are orthogonal. To see this, note that if such a rectangular wall semipermeable to a particles were to isothermally move from the left side of a box (of volume V) to the middle, leaving the $N/2$ particles of state a untouched but shifting all $N/2$ of the b particles to the right side, then heat $NkT/2 \ln(2)$ would be rejected to the external

bath (since the volume of the $N/2$ *b* particles is isothermally compressed by a factor of 1/2). Compressing the left side (containing only *a*) to volume $V/4$, and expanding the right side to volume $3V/4$, we reject an amount of heat $NkT/4 \ln(2) - 3NkT/4 \ln(3/2)$ to the reservoir, ending up with a total heat-bath entropy increase of $3Nk/4 \ln(4/3)$. Comparing this to the change in entropy of the system of gases, via the density matrix calculation, we find that total entropy is conserved overall if and only if $\langle a|b\rangle = 0$ (Neu55).

Finally we make a few comments on *measurements*, which are the most arcane and yet the most elementary operations which can occur in quantum mechanics. Measurements are not thermodynamically reversible, in general, and indeed in quantum computation this imposes an additional cost on computation at its most elementary level. On one hand it is a difficult subject to analyze, leading even people like von Neumann to talk about

> "psycho-physical parallelism. . . it must be possible so to describe the extra-physical process of the subjective perception as if it were in reality in the physical world – i.e., to assign to its parts equivalent physical processes in the objective environment, in ordinary space…no matter how far we calculate, at some time we must say: and this is perceived by the observer." (Neu55, p. 419, English translation)

To this day philosophers still debate the connections between consciousness, quantum mechanics, and measurement, often incorrectly (Cha97). We will not attempt to repeat this mistake here.

I.1.ii. Information theory

Information theory began ostensibly as a tool for analyzing losses in telephone networks, but quickly attained the air of something fundamental (Sha48) (Sha49): anything describable by a random variable *x* with probability distribution $p(x)$, can be associated with a single number describing its importance and uniqueness, $H(p)$, given by

$$-\sum_x p(x)\log_2 p(x) = \langle \log_2 p(x)\rangle$$

where the base-2 logarithm indicates that our information is being measured in *bits* (Cov91), and the symbol $\langle y(x)\rangle$ represents the mean of $y(x)$ over the probability distribution $p(x)$. It is possible to show that $H(p) \equiv H(p(x_1), p(x_2), \ldots p(x_n))$ is the only functional of $p(x)$ which is invariant under permutations of the x_i (as expected for a function whose argument has the structure of a set), and which satisfies the 'objectivity requirement' that

$$H(p(x_1), p(x_2), \ldots p(x_n)) = H(p(x_1) + p(x_2), \ldots p(x_n)) + (p(x_1) + p(x_2)) \times H\left(\frac{p(x_1)}{p(x_1)+p(x_2)}, \frac{p(x_2)}{p(x_1)+p(x_2)}\right)$$

which says that the information gotten by examining the value of a random variable can be obtained by first determining whether it is in a certain set of values, then determining what member of the set it corresponds to (Fuc96). Note that applying the objectivity requirement to an entropy function of a *n*-valued probability distribution gives the entropy in terms of (*n*-1)-valued probability distributions – in particular, repeatedly applying the objectivity requirement gives $H(p)$ expressed entirely in terms of the function $H(q, 1\text{-}q) \equiv f(q)$, where *q* is some real number

between 0 and 1, inclusive. Therefore to find the unique functional $H(p)$, all we need to do is solve for $f(q)$.

Finding $f(q)$ is fairly simple (Dar70): let us introduce a parameter a that we will later let go to the limit of 1, and modify the objectivity requirement to read

$$H_a(p(x_1), p(x_2), \ldots p(x_n)) = H_a(p(x_1) + p(x_2), \ldots p(x_n)) + (p(x_1) + p(x_2))^a \times H_a(\frac{p(x_1)}{p(x_1) + p(x_2)}, \frac{p(x_2)}{p(x_1) + p(x_2)})$$

Then from the permutation rule, $H_a(x, y, 1\text{-}x\text{-}y) = H_a(y, x, 1\text{-}x\text{-}y)$, and applying the modified objectivity requirement gives

$$f(x) + (1\text{-}x)^a f(y/(1\text{-}x)) = f(y) + (1\text{-}y)^a f(x/(1\text{-}y)),$$

or, if we let $q = y/(1\text{-}x) = y/p$,

$$f(p) + p^a f(q) = f(pq) + (1\text{-}pq)^a f(\ (1\text{-}p)/(1\text{-}pq)\ ).$$

With a little algebra, this last equation can be used to show that $F(p, q) = f(p) + (p^a + (1\text{-}p)^a)\ f(q)$ is symmetric in p and q. Setting $F(p, 1/2) = F(1/2, p)$ then gives

$$f(p) = (2^{1\text{-}a} - 1)^{-1}(p^a + (1\text{-}p)^a - 1),$$

where we are taking $f(1/2)$ to be 1, as a normalization. This gives, upon inserting the definition of $f(p)$ back into the definition of $H(p)$, we find thta

$$H(p(x)) = (2^{1\text{-}a} - 1)^{-1}(\sum_{i=1}^{n} p_i^a - 1)$$

and taking $a \to 1$ using L'Hopital's rule instantly gives

$$H(p) = -\sum_x p(x) \log_2 p(x),$$

as desired. We have now come up with two justifications of the definition of the entropy – one from quantum mechanics, and one from the simple properties of symmetric functions. The definitions are, interestingly, identical. The mathematical properties of the entropy function in various situations are no less profound.

Shannon's noiseless coding theorem asserts that if messages a_i occur with probabilities $p(a_i)$, then for long strings of messages, the mean minimal number of bits l_{min} needed to represent each message satisfies $H(p) \le l_{min} \le H(p) + 1$. The proof is given in (Cov91), and is tightly coupled with two other theorems that appear in that text, namely the *Kraft inequality* and the *asymptotic equipartition property*. We will not discuss these two auxiliary constructs here, but we will give the proof of the noiseless coding theorem, since we will later want to consider the quantum version.

The proof is based on the weak law of large numbers (WLLN), which says that for a set of IID (independent, identically distributed) random variables x_i with finite variance,

$$\forall(\delta, \varepsilon > 0)\exists N \forall(n > N) P(|\frac{1}{n}\sum_{i=1}^{n} x_i - \bar{x}\ | > \delta) < \varepsilon\ .$$

This is a fancy way of saying that by taking more and more samples of the probability distribution, we can make their average arbitrarily close to the exact mean of the distribution. The WLLN can be proven by representing the probability distribution of the sum of N independent identically distributed random variables as a convolution of N copies of the

individual probability distribution (Dra67). Taking $x_i = -\log(p(a_i))$, where a_i are the messages from a source A (which emits message a_i with probability $p(a_i)$), we find that $\bar{x}$ equals $H(A)$. We may note that a sequence a of such messages $a_1 \ldots a_N$ satisfies $P(a) = p(a_1)...p(a_N)$, and we define the set of sequences a satisfying

$$|\frac{-1}{N}\log(P(a)) - H(A)| \leq \delta .$$

as the set of *likely sequences*. It immediately follows that

$$2^{-N(H(A)-\delta)} \geq P(a) \geq 2^{-N(H(A)+\delta)}$$

and therefore the number of likely N-message sequences goes as $2^{NH(A)}$, while the fraction of N-message sequences that are 'unlikely sequences' goes to zero as N increases. Noting that $N(H(A)+\delta)$ bits is enough to encode all the likely sequences of N messages, we have shown the first part of the noiseless coding theorem, which is equivalent to the statement that if $H(A)+\delta$ bits are available for each message, sufficiently long sequences of messages from A can be coded into binary with probability of error less than ε, for any ε greater than zero. Conversely if only $H(A) - \delta$ bits are available for each message, then for sufficiently long sequences of messages, the probability of error will be greater than $1-\varepsilon$, for any ε greater than zero. One can see this simply by noting that $N(H(A) - \delta)$ bits is only enough to encode $2^{N(H(A) - \delta)}$ sequences, which is a fraction $2^{N(H(A) - \delta)} / 2^{NH(A)} = 2^{-N\delta}$ of the likely sequences. This fraction clearly becomes arbitrarily small as the message length increases.

We can define the *relative information*, $H(x \mid y)$, as

$$\sum_x p(x) \sum_y p(y \mid x) \log p(y \mid x),$$

and it is clear that if $H(x, y)$ describes the entropy of the joint distribution of x and y, then

$$H(x, y) = H(x) + H(y \mid x).$$

The *Kullback-Liebler divergence*, or *relative entropy*, measures the cost of assuming that a variable described by $p(x)$ is actually given by $q(x)$ (Cov91). This cost is nicely described as that of "keeping the expert honest" in (Fuc96). It is given by the equation

$$D(p \mid q) = \sum_{x,y} p(x) \log \frac{p(x)}{q(x)}.$$

Note that it is not a metric, since the distance between p and q depends on which order you pass the arguments to the D function.

The *mutual information* between probability distributions $p(x)$ and $p(y)$ corresponds to the information that one random variable contains about another, and is given by

$$I(x, y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)\,p(y)} = D(p(x, y) \mid p(x)\,p(y))$$

It is symmetric in its arguments, and equals $H(x) - H(x \mid y)$. From these simple definitions, the properties of the logarithmic function, and a few simple tricks like Jensen's inequality, one can derive some very interesting properties of information-theoretic function, and learn how to apply them to models of communication channels, data encryption, data compression, Markov processes, maximum-rate coding, and coding strategies that are resilient to noise (Cov91) (Wie).

We briefly comment on methods of telling probability distributions apart, a crucial topic when considering the final step of any quantum computation: namely, the interpretation of the signal as a collapsed signature of the quantum state. The first method uses *Bayes' rule*: consider two probability distributions $p(x)$ and $q(x)$ for a random variable x which takes any one of n distinct values, and which occur in nature with prior probabilties P and Q respectively. If one is given a sample, x_0, and asked to guess which probability distribution generated it, then one may try to minimize the probability of guessing wrongly: the resultant best guess is $p(x)$ if $P\ p(x_0) > Q\ q(x_0)$, and $q(x)$ otherwise (Dra67) (Opp). For N samplings, we must turn to the *Chernoff bound*, which says that the probability of guessing wrongly is less than λ^N, where λ is the minimum value of

$$\sum_{i=1}^{n} p(x_i)^a\, q(x_i)^{1-a}$$

over all possible $a \in [0,1]$. The proof, which is quoted in (Fuc96), follows from two simple observations: (1) Bayes' decision rule can be used to express the probability of guessing wrongly in terms of the minimum of $P\ p(x_0)\,p(x_1)...p(x_N)$ and $Q\,q(x_0)q(x_1)...q(x_N)$, and (2) the minimum of two positive numbers c, d satisfies

$$\min(c,d) \leq c^a d^{1-a},$$

for any $a \in [0,1]$. The value λ is expressible as a KL divergence, which can help make the Chernoff bound more intuitive.

We will also consider the *fidelity*, which plays a significant role in discussions about quantum information theory, especially in the context of quantum channels. The classical fidelity, for the two probability distributions defined above, is given by

$$F(p(x), q(x)) = \sum_{i=1}^{n} \sqrt{p(x_i) q(x_i)}\,.$$

This looks a lot like a dot product magnitude in vector geometry. It is noted (Fuc96) (Jef) that the arccosine of this quantity is the geodesic distance between $p(x)$ and $q(x)$ in probability-space, with the *Fisher information metric*

$$ds^2 = \sum_{i=1}^{n} \frac{(dp(x_i))^2}{p(x_i)}\,.$$

as the Riemannian metric for this space. This quantity has been associated with interesting differential-geometric properties of probability distributions. In particular, the expectation-maximization algorithm decreases the distance between the true and estimated probability distributions, when the distance is measured with respect to the Fisher information (Mur). It is not clear what use these methods have in real life, but they are an interesting formalism which provides yet another picture of the universe, and one that is especially appealing to the machine-learning, artificial intelligence, and neurocomputation communities.

I.1.iii. Computability

We begin with classical models of computation. Computation is notoriously difficult to define; it gladdened many people when the definitions of Turing, Kleene, Church, and many other early

pioneers in the field were all proven equivalent. This definition is appreciated for its comprehensibility, aesthetic value, and consistency with all known classical, anthropically interpretable, physically realizable means of computation. We follow Sipser (Sip97). The vast bulk of computation theory is arrived at through meditation and debate; many people are now trying to embed it in a more physical context.

I.1.iii.a. Automata Theory

The simplest interesting computer is a finite state machine (FSM), or *finite automaton* (FA). All realizable computers are essentially FAs, although relaxing the constraint of finiteness allows the discipline of computability to take on deeper meaning. A FA is defined by a finite set of states Q, one of which is designated the start state q_0, and a subset of which comprises final states F (called *accept* states), as well as a finite *alphabet* Σ of symbols and a *transition function* $Q \times \Sigma \rightarrow Q$ which selects the next state of the machine, given the current state and one symbol from an input string. FAs are very useful constructs for practical computer architectures; often a significant portion of the intellectual effort and cost related to a complicated microprocessor (e.g., with pipelining, branch prediction, reorder buffers, and much more) is used to form very complicated finite state machines for control (Hen).

Given an input string w of symbols in Σ, we say that a FA *accepts* w if it reaches a state in F in some finite time; the set of states that the FA passes through is called the *computation history*. A FA M *recognizes* a *language* A if $A = \{w \mid M \text{ accepts } w\}$; such a language is called *regular*. Given any two languages A, B, we can treat them as sets, and define the union $A \cup B$, concatenation $A \circ B = \{xy \mid x \in A,\ y \in B\}$, and the star $A^* = A_0 \cup A_0 \circ A_0 \cup A_0 \circ A_0 \circ A_0 \cup \ldots$, where $A_0 = A \cup \{\varepsilon\}$ and ε denotes the empty string. Interestingly, the class of regular languages is closed under these three constructions; the proof of this claim for the union operator is easily done by constructing a new machine which operates on a set of states which is the Cartesian product of the sets of states for the FAs which recognize the two component languages. The other two proofs require a new formalism, *nondeterministic* computation. Symmetry arguments suggest one further observation – the complement of a regular language is always a regular language (just swap the accept and non-accept states in a DFA), and therefore by DeMorgan's laws, regular languages are closed under intersection.

A *nondeterministic finite automaton* (NFA) can make several choices at each state, depending on the input, and follow them all in parallel paths; if any path halts in a final state, the machine is said to accept. Nondeterminism is key to many important concepts in theoretical computer science, including the famous definition of NP-completeness (which we discuss later). Formally the only qualifier we must add to our definition of FA to make it compatible with NFAs is that the transition function produces, from Q and a letter, a set of possible next states, called the *power set* $P(Q)$ of Q, for reasons that will be clear to anyone who is familiar with set theory. NFA computation follows each element of this power set in parallel, forming a tree structure. We will call the ordinary FAs DFAs, which stand for *deterministic finite automata*, for clarity.

Interestingly, every NFA has an equivalent DFA; one simply simulates the k states of the NFA with a DFA with 2^k states, each one corresponding to a possible subset of the states of the NFA.

(Note the exponential correspondence between the resources required for deterministic and nondeterministic computation; this will come back to haunt us again and again, especially in the context of quantum computing!) Therefore the set of languages recognized by NFAs is the same as the set of regular languages, but we can use this more powerful idea to our advantage. For the proof of the closure of regular languages under the concatenation operator, suppose that one has languages *A*, *B* which are recognized respectively by NFAs *M*, *N*: simply connect each final state of *M* to the start state of *N*, with transition functions conditioning on the empty string. For the proof of closure under the star operator, simply connect the final states of a NFA to its start state, conditioning on the empty string. These proofs are elegant and simple.

A *quantum finite automaton* (Amb98) can be considered to be directly analogous to the DFA which is in a superposition of states; it is said to recognize a language *L with probability* $1/2+\varepsilon$ if it accepts/rejects all words in/outside *L* with this probability. We do not discuss these QFAs much further; although there are some interesting debates over their power, they are abstract and not terribly useful.

Anyone who has used Unix knows about *regular expressions*, which essentially are strings recursively built up of elements of an alphabet Σ, the empty string ε and the empty language ϕ, and the 3 symbols $\cup$, $\circ$, and *. For example, the language of decimal numbers could be written as $\{+,-,\varepsilon\}(ZZ^* \cup ZZ^*.Z^* \cup Z^*.ZZ^*)$, where $Z = [0..9]$. Amazingly, a language is regular iff it can be described by a regular expression! The nonintuitive part of the proof is to show that a regular language can be described by a regular expression: it is surprisingly simple to convert a DFA into a regular expression by considering generalized nondeterministic finite automata (GNFA), for which *any regular expression* (not just an individual input symbol) can serve as the condition for the transition function. One begins with a GNFA which is just the DFA of interest, then recursively combining the states of the GNFA until there are only two left (the start and accept states), which are connected by a single regular expression, which is what we desired. Note the interesting corollary that any GNFA is equivalent to a GNFA with only two states.

Nonregular languages are very common, for example the set of strings with equal numbers of 0s and 1s. The *pumping lemma* is a useful tool for proving languages nonregular: for any regular language *A*, there exists a number *p* (a constant for all strings in A) such that any string *s* with length $|s| \geq p$ can be written as *xyz* for some substrings *x*, *y*, and *z*, such that $xy^*z \in A$, $y \neq \varepsilon$, and $|xy| \leq p$ (Proof: since DFAs are Markovian, if a state is repeated twice, then the string in between the repeated states can be replicated arbitrarily many times, and still yield a valid string of the language; by the pigeonhole principle, it is clear that a state is repeated twice if $p \geq |Q|$). Assuming that a *p* exists for a language *A*, one can then show *A* to be nonregular by exhibiting a string *s*(*p*) which cannot be divided into substrings as claimed by the pumping lemma. The pumping lemma focuses on locality – the fact that in a certain string, the DFA is always confined to monitoring behavior over a certain little area. Languages that are easily proved to be nonregular using this result include $\{0^n1^n\}$, $\{ww \mid \text{all binary strings } w\}$, and $\{0^m1^n \mid n>m\}$.

It is possible to convert a DFA into a *minimal DFA* – that is, the DFA with the least number of states, that recognizes a particular language – in polynomial time. The algorithm works by consolidating the DFA into a regular expression, and then systematically writing a DFA that

corresponds to the resultant regular expression. A hierarchy can be constructed based on minimal DFAs: for every *k*, there exists a language recognized by a DFA with *k* states, but not by one with *k* -1 states.

A more complex language architecture is the *context-free grammar* (CFG), which possesses a more recursive flavor than the mere repetition and retracing which characterizes regularity. Compiler aids like *bison* and *yacc* can generate parsers from context-free grammars; indeed, CFGs are complex enough to describe natural languages like English (Edw87). A CFG consists of a list of substitution rules $A_i \rightarrow B_i$, where A_i is a variable and B_i is a string of variables and terminals (so called because they cannot be replaced by anything else – e.g., they can't appear on the left-hand side of a rule). The computation history which lists the substitutions that one makes, in any computation involving a CFG, is called a *derivation*, much like proofs are derived in formal logic: possible derivations are easily pictured using trees. The strings which appear on the final lines of derivations make up a *context-free language*, or CFL. Any compiler which hopes to process a CFG must proceed via analytical recursion. One must often define arbitrary precedence relations for rules, since rules can be *ambiguous* – for example, does $a+a\times a = a+a^2$ or $2a^2$?

We say that a CFG is in *Chomsky normal form* (CNF) if every rule is of the form A→BC or A→a (where A, B, C are variables and a is a terminal). Interestingly, every CFL is generated by a CFG which can be written in CNF; the proof consists of just cleaning up all the rules by introducing intermediate variables for complicated rules and throwing away trivial rules. A length-*n* string requires exactly 2*n*-1 steps in a derivation from a CFG in CNF; conversely, if a CFG *A* in CNF contains *n* variables and *A* generates a string with a *m*-step derivation, where *m*>*n*, then the CFL generated by *A* is infinite.

Another automaton is the *pushdown automaton* (PA), which is essentially a simple nondeterministic stack machine; PAs are equipotent to CFGs. The stack is essentially an infinite LIFO device; the stack may have its own alphabet Γ and the transition function maps $Q\times\Sigma\times\Gamma$ to the power set of $Q\times\Gamma$. One example of a language which a PA can recognize but a NFA cannot is $\{a^i b^j c^k \mid i, j, k > 0, (i = j) \vee (i = k)\}$; for this language (and many other languages), it is possible to see that nondeterminism is essential for its recognition (a deterministic pushdown automaton could not recognize this language). It can be proved that a language is a CFL iff it is recognized by a PA (to convert a CFL *A* to a PA, create a nondeterministic PA that forever guesses derivations for an input string $w \in A$, halting if it ever generates *w*; to convert a PA *P* to a CFG *G* that generates strings accepted by *P*, create *G* with variables A_{pq} so that A_{pq} generates all strings that take state *p* to state *q*, and show that A_{pq} generates *x* iff *x* takes *P* from *p* to *q*, using induction. Whew!). Trivially, we also find that regular languages are context free, since they are recognized by DFAs (which are just PAs that ignore the stack). In fact, regular languages and CFLs share a lot of structure: one can prove a variant of the *pumping lemma* for CFLs; for any CFL *A*, there exists a number *p* (a constant for all strings in *A*) such that any string $s \in A$ can be divided up into 5 substrings *uvxyz*, such that $\forall i(uv^i xy^i z \in A)$, $|vy| > 0$, and $|vxy| \leq p$ (proof by finding repeated variable symbols as before, using the pigeonhole principle; the factor 5 comes very simply from the tree structure; *v* and *y* represent the tree branches closest to the main trunk, which can therefore be repeated ad infinitum). With the pumping lemma for CFLs, one can for

example prove that $\{a^n b^n c^n \mid n > 0\}$ is not a CFL, although the case-by-case analysis for 5 substrings can be tricky. Other languages that are not CFLs include $\{a^i b^j c^k \mid i<j<k\}$, $\{ww \mid$ all binary strings $w\}$ (to see this, pump using $0^p 1^p 0^p 1^p$), and $\{w\#x \mid w$ a substring of $x\}$. The intersection of two CFLs is not always a CFL; therefore the complement of a CFL need not be a CFL (although CFLs are closed under the regular-language operations union, concatenation, and star). Not surprisingly, the intersection of a regular language and a CFL is also a CFL. Finally, we mention that nondeterministic PAs are strictly more powerful than deterministic PAs – as we noted at the beginning of this paragraph.

A note on *quantum finite-state* and *push-down automata*: one can define a quantum grammar by summing over all the derivations of a word to find the amplitude of a word; the standard proofs carry over almost identically for a great many formalisms in the theory of computation; see (Moo97) for a detailed analysis. It is not yet clear how this picture enhances our understanding of the computational power of quantum systems.

I.1.iii.b. Turing Machines

What happens if you create a deterministic PA with two stacks? Then locality is not an issue, since you can join the two stacks at the top into a single infinite list, and simulate pushing and popping with a read/write head that moves left or right. The result is a *Turing machine* (TM), the most powerful classical automata which can be physically realized, according to *the Church-Turing thesis*. A TM is an infinite tape with a read/write head, with a specific finite state machine controlling the behavior of the R/W head (i.e., moving to the left or right, and reading or writing to the tape). The transition function therefore maps the current configuration $Q \times \Gamma$ to a new one, $Q \times \Gamma \times \{L, R\}$, i.e. the head reads the tape and then, based on the current state, changes the state and the tape, and then moves left or right. Some conventions are useful: the symbol $u \in \Gamma$ is used to denote a blank entry on the tape; there are 3 special states in the finite state machine, called *start*, *accept*, and *reject*. If a TM M halts in the accepting state upon input w, we say that M accepts w. The set of such w is called the *language* accepted by M. (The definitions of language, accepting, rejecting, and so on are all fairly standard throughout computation theory – indeed, they form the basis for comparing paradigms of different computational power.) A language L is *Turing-recognizable* (*recursively enumerable*) if a TM M recognizes it – e.g., M halts in the accept state when the input is an element of L. However, M is not required to halt at all upon inputs that are not in L. A language L is *decidable* (*recursive*) if a TM M decides it – e.g., it halts in the accept state when the input is an element of L, and it halts in the reject state when the input is not in L. L is *co-Turing-recognizable* if it is the complement of a Turing-recognizable language. It is clear that "L is decidable" means the same thing as "L is Turing-recognizable and co-Turing-recognizable."

Justification for the Church-Turing thesis comes mostly from detailed yet abstract omphaloskepsis. k-tape Turing machines can be simulated by a 1-tape TM in order n^2 time, where n is the number of filled cells on all the tapes at a particular moment in time, and a nondeterministic Turing machine (NTM) can be simulated by a 3-tape (one for input, one for simulation notes, and one for the actual output) TM in 2^n time, where n is the number of steps in each branch of the NTM's computation, and breadth-first search is used for the simulation (so

that every branch of the computation gets some running time, but non-terminating branches don't cause the TM to enter an infinite loop). A TM where you can only write on each square of the tape once, is also equivalent to an ordinary TM. However, a Turing machine that cannot write on the part of the tape containing the input string can only recognize regular languages! Nobody has ever come up with a realistic computation scheme which could solve a problem that a TM couldn't. (The efficiency with which a certain implementation solves a problem, on the other hand, is a separate issue that we will consider in the next section.)

Certain problems are *undecidable*. There is no general way to tell whether an exponential Diophantine equation has an integer root, for example (Hilbert's 10^{th} problem), although certainly the language of such equations is recognizable (just start testing roots, and if the equation really is solvable, then eventually one set will work). It is clear that a language is decidable iff it can be enumerated (e.g., written down in a list) in lexicographic (e.g., alphabetical) order. In particular, the following languages are decidable (the finite state machine proofs are easily accomplished by construction of the appropriate TM and applying the pigeonhole principle to repeated states):

- $\{\langle B, w\rangle \mid B$ is a DFA (or NFA) that accepts string $w\}$,
- $\{\langle R, w\rangle \mid w$ is generated by the regular expression $R\}$,
- $\{\langle A\rangle \mid A$ is a DFA and $L(A)$ is empty$\}$,
- $\{\langle G, w\rangle \mid w$ is generated by the CFG $G\}$(check all $(2n-1)$– step derivations of the grammar after converting to CNF),
- $\{\langle G\rangle \mid G$ is a CFG and $L(G)$ is empty$\}$ (work backwards from the terminals, marking the left hand side of an $\rightarrow$ equation whenever all the symbols on the right hand has been marked, until no more progress can be made),
- all CFLs (for each CFL A and input w, just run the simulator two lines up on $\langle G_A, w\rangle$).

This is an impressive list. But there are many undecidable languages, unpredictable dynamical systems, and unverifiable processes, and this is fundamental to the theory of computability. It is impossible to decide the language $L_{UTM} = \{\langle M, w\rangle \mid$ TM M accepts string $w\}$; L_{UTM} is said to be recognized by the universal Turing machine *UTM*, which simulates the action of an arbitrary TM on an arbitrary input, and accepts if the simulated TM accepts. The proof is by diagonalization. One way to see this is to simply note that the set of all languages can be put into corresepondence with the elements of the real line, but the set of TMs is countable, since each can be written down as a program. Another way to see this is to exhibit an undecidable program: define a program $D(\langle M\rangle)$ which returns the opposite of $UTM(\langle M, \langle M\rangle\rangle)$, and obtain a contradiction by considering the value of $D(\langle D\rangle)$. Interestingly, given a Turing-recognizable language A_1 consisting of descriptions of arbitrary TMs, there exists a decidable language A_2 containing TMs equivalent to those in A_1.

The primary tool for proving that a problem is undecidable is by *reduction*: assume that the problem is solvable, and then show that it solves some other, undecidable problem. For example, it is easy to see that the decidability of L_{UTM} reduces to $\{\langle M, w\rangle \mid M$ halts on $w\}$, $\{\langle M\rangle \mid M$ is a TM and $L(M)$ empty$\}$, and $\{\langle M\rangle \mid M$ is a TM and $L(M)$ is regular $\}$: in fact, according to Rice's theorem, testing any property of the Turing-recognizable languages is undecidable – including whether or not a universal TM ever enters a certain state, whether or not a universal TM ever attempts to move off the tape, and so on. Formally, given any language of programs P such that

the question '$M \in P$?' can be answered only by knowing the language accepted by M, and such that P doesn't contain every TM, P is undecidable.

Another tool for proving undecidability for certain 'existence problems' is the method of *computation histories* (CH), or complete records of the intermediate states that the TM passes through while it computes. An *accepting CH* is a sequence of configurations $C_1,\ldots C_n$ where C_n is an accepting configuration and C_i follows from C_{i-1}; a NTM's CH may look like a tree, with one leaf in an accepting configuration. A *linear bounded automaton* (LBA) is a TM whose head is restricted to the part of the tape containing the input string; even if the TM alphabet is larger than the input string alphabet, the memory of the TM is still required to be linear in the input. The deciders for DFAs and CFGs are LBAs, for example, and {$<M,w>$ | M is a LBA that accepts w } is decidable. To see this last point: it is clear that if an LBA has q states and g symbols in the tape alphabet, there are exactly qng^n configurations for an input of length n; this shows that all LBAs are decidable (since the number of configurations is finite, one simply has to run the LBA for at most qng^n steps to see if it halts or loops). However, {$<M>$ | M is an LBA and $L(M)$ is empty} is undecidable (proof by reduction from *UTM*: one can run a decider for the emptiness of LBAs on an LBA which checks the CHs for $<M,w>$, which then solves the halting problem) (Sipnote).

One concept that is familiar from Myhill's theorem (Har) is *many-one reducibility*: for languages A, B, $A \leq_m B$ if there exists a computable function f such that $w \in A \leftrightarrow f(w) \in B$. That is, the problem A is many-one reducible to the problem B if there exists a computable function that, given a method of solving B, enables us to solve A; therefore if $A \leq_m B$, and B is decidable, A is decidable; we have often applied the contrapositive of this result in the preceding discussion. Similarly, if $A \leq_m B$ and A is not Turing-recognizable, then neither is B (setting A to be co-*UTM*, the complement of *UTM*, and therefore a non-Turing-recognizable set, is a typical start of proofs using this result).

Recursion theory is an even more abstract formalism for dealing with computability theory, and contains many surprises. We will not go into this very deeply, but there is a nice book (Har) for those who are interested.

Self-printing programs are a curiosity in many languages. For example, a clever example in C is

```
main(a){a="main(a){a=%c%s%c;printf(a,34,a,34);}";printf(a,34,a,34);}
```

which prints itself on machines which use ASCII. Importantly, there is a computable function $f(w)$ such that for any string w, $f(w)$ is the description of the Turing machine P_w that prints out w, then stops. Formally, one can state the r*ecursion theorem*: for a TM T that computes $t(a,b)$, there exists a TM R that computes $r(a)$ so that $r(w) = t(<R>,w)$ – that is, this TM can obtain its own description, then go on to compute with it; the proof is similar to the self-printing TM proof. This version of self-reference is almost a diagonal argument in itself, and gives another proof of the undecidability of UTM.

Also, one can show that *MIN* = { $<M>$ | M is a minimal TM, e.g. the shortest TM that generates its particular output } is not Turing-recognizable (proof: by the recursion theorem, assume that a program A enumerates *MIN* and runs the enumerated program on an input: eventually a minimal program longer than A will be generated; but this is a contradiction, since A would then compute

the output of a minimal program longer than *A*!) (Cha96). The fixed-point theorem of recursion theory says that for any computable function $t(a)$, there exists some $t(\langle F \rangle)$ that describes a Turing machine equivalent to *F*; the proof is trivial using the recursion theorem (*F* = "on input *w*, obtain own description, compute $t(\langle F \rangle)$ to get a TM *G*, run *G* on *w*" – then *F* simulates *G*, so $\langle F \rangle$ and $t(\langle F \rangle)=\langle G \rangle$ are equivalent), but entirely nonintuitive, in the phraseology of traditional recursion theory (Sac).

I.1.iv. Complexity

Complexity theory measures the resources that a computation demands. The time complexity of an algorithm is one of the most common determinants of the success of a computational strategy, although occasionally one finds that exorbitant amounts of energy or incredible numbers of particles can be substituted for a long amount of time. For complicated problems with varying strategies, often the *worst-case* and *average-case* strategies will be vastly different – as anyone with experience with algorithms such as the Quicksort algorithm can attest. We say that a program (or Turing machine) with input *n* is an $O(f(n))$ program if, to leading order, the time that the program takes to execute (or for the Turing machine to halt) is asymptotically $f(n)$ *or less*, up to a constant factor. This implies that $O(n) = O(2n) = O(2n+1550)$, for example. (The notation $o(f(n))$ means asymptotically strictly less than $f(n)$, and sometimes $\Omega(f(n))$ is used to mean asymptotically greater than or equal to $f(n)$. Asymptotic equality is denoted by $\theta(f(n))$.)

The time *complexity class* $O(t(n))$ is the set of languages decided by $O(t(n))$ Turing machines. For most problems which are posed in a positive fashion, it is possible to construct a simpler program which *verifies* that an input is accepted when a suitable symbolic proof (or *certificate*) is included. Determining the complexity class of a language is difficult because of the negative nature of this definition: one must show that no one can make a faster program, that decides the same language. (One can show that determining the complexity class of a language is, in general, uncomputable – see (Cha96) for details.) Also, the asymptotic bound on the complexity of a function may be absolutely useless in a practical sense, if the program is only to be run on small values; the value of this categorization to working computer programmers is often not clear.

However, certain classes have proven very useful in theoretical computer science, the most basic of which is the class P, the class of programs which run in time polynomial in the input. Simple examples of class P problems include determining whether there is a path between two nodes in a directed graph (use breadth-first search), the Euclidean algorithm for integer division, and deciding any CFL (use dynamic programming, which gives an $O(n^3)$ solution). Most simple, well-defined problem-solving strategies are in P. Significantly, it is believed in some circles that any *classical* universal TM can simulate another classical universal TM in polynomial time; this is known as the *strong Church-Turing thesis*. Algorithms which take exponential time are usually considered to be intractable; this includes most searches and brute-force algorithms (e.g., complex, poorly-defined strategies).

An interesting note is that these results depend on the fact that we encode our problem as a string on a tape, and then run a Turing machine head down the tape. Any search or other computation involving a global property of the string will demand that we visit every cell on the tape at least once – this gives a lower bound on the complexity of the problem. What if we encoded the

problem as a quantum system? It is now known that for some problems, this encoding can make the problem more readily solvable.

Some theorems are in order (Hop79): given an *n*-tape Turing machine (TM) that runs in $t(n)$ time, there is a single-tape TM that runs the program in $O(t(n)^2)$ time. Given a nondeterministic Turing machine (NTM) that halts in time $t(n)$, there is a single tape TM that runs the program in $2^{O(t(n))}$ time. (Recall that for an NTM, the halting time is the length of the longest branch of the tree-representation of the possible paths the program can take. This is the reason that nondeterministic problems, or general inverse/search problems, take exponential time – all possible trees have to be explored, and the number of nodes in such a tree is exponential in its depth.)

There is a class of problems known as NP problems, which seem to require exponential-time (brute-force-style) algorithms, but nobody has proven that they cannot be solved in polynomial time. All of these problems can be verified in polynomial-time, and are decided by nondeterministic polynomial time TMs. Indeed, these two conditions are equivalent, and either is commonly used as a definition of NP. One can think of the two conditions as inverse criteria, related by the concept of search, as suggested by the previous paragraph. Well-known NP problems include determining whether a directed graph contains a path between 2 nodes which passes through all the nodes (a Hamiltonian path), determining whether a graph contains a complete subgraph of a specified size (the clique problem), and determining whether a subset of a given set of numbers adds up to a given integer (the subset/sum problem). It is easy enough to identify certificates for these problems: for the Hamiltonian path problem, for example, it is sufficient to create a certificate containing the vertices of the path in the order of traversal. The set of the complements of the languages in the complexity class NP is called coNP, but due to the negative-requirement nature of the sets of coNP, it is not known how to find useful verifiers for these problems – it is not even known whether NP = coNP. (As noted earlier, negative statements are difficult to satisfy because they often require refutation of an infinite number of possible statements. Compare this to the fact that negative definition such as compactness, uncountability, logical completeness, and nonhomeomorphic require complicated, nonconstructive ideas, often using proofs by contradiction. There are nice parallels between positive and negative statements, forward and inverse problems, determinism and nondeterminism, construction and proof-by-contradiction, and assertion and search.) Surprisingly, it has never been proven that NP is different from P: in other words, no one has ever found a way to replace brute force searching, or to show that searching is an essential tool, for solving problems that can be verified in polynomial time.

A useful tool in complexity theory is to construct subclasses of complicated classes. For example, the existence of a polynomial-time algorithm for any *NP-complete* language is equivalent to the amazing statement P = NP. We say that $A \leq_P B$, or language A is *polynomial time reducible* to B, if there exists a function f which executes in polynomial time such that $w \in A \leftrightarrow f(w) \in B$ – i.e., A can be decided in polynomial time if B can be decided in polynomial time. Therefore another way to say that an NP language B is NP-complete is to say that for all A in NP, $A \leq_P B$. It is rather amazing at first glance that any NP-complete problems exist; one such problem is SAT, the problem of determining whether a set of Boolean variables can be assigned such that a certain well-formed-formula is true. The proof of the fact that SAT is NP-complete,

the Cook-Levin theorem, associates a Boolean formula with each computation deciding whether $w \in A$, such that the TM accepts w only if the Boolean formula is satisfiable. Intuitively, this is not surprising since a digital computer performs similar operations whenever it simulates a Turing machine using its logical gates. The proof, however, is intense, detailed, and well-known, so we omit it here. A list of NP-complete problems is a useful reference, especially for the aspiring quantum-computation theoretician seeking to find some subtle property of quantum mechanics that allows NP-complete problems to be solved in shorter amounts of time. (Indeed, finding an NP-complete problem that can be solved by a quantum computer in polynomial time is one of the holy grails of quantum computing research.) Since the polynomial-time reducibility constructions can be very contrived and tricky, proof strategies are briefly listed in parentheses – the interested reader can refer to (Sip97), (Hop79), or (Cor90) for details:

- 3SAT, the problem of determining whether a formula in conjunctive normal form with precisely 3 literals per disjunctive block is satisfiable;
- the clique problem (3SAT $\leq_P$ clique);
- the vertex covering problem, the problem of finding the smallest set of nodes of a graph that collectively touch every edge (3SAT $\leq_P$ vertex-covering);
- the Hamiltonian-path problem (3SAT $\leq_P$ Hamiltonian-path), with directed or undirected graphs;
- the subset/sum problem (3SAT $\leq_P$ subset/sum);
- the pseudo-halting problem that determines whether a certain NTM accepts an input in t steps or less;
- the coloring problem, which determines whether a graph can be colored by 3 colors so that no two adjacent nodes have the same color (or, whether one can color a graph on m nodes with b colors, so that no two identically colored vertices are connected by an edge);
- the traveling salesman problem, which finds the minimum-length path that a salesman would take to visit all of the cities on a map exactly once.

Other classes exist, and a working familiarity with their names is useful for someone trying to master the field: an *NP-hard* language A is such that all NP languages are polynomial-time reducible to A (even though A may not be NP). We later discuss why nonlinear quantum mechanics (which does not exist, as far as we know, but might emerge from a coherent quantum theory of a nonrenormalizable force like gravity) implies that NP-complete problems could be solved in polynomial time (Jac) (Abr98) (Free).

We briefly remark on methods of solving NP-complete structured search problems. We can describe these problems as *constraint satisfaction problems* (CSPs) in that a set of variables need to be assigned values while satisfying some set of relations. While a solution may be very hard to find, it is often easy to find common elements to non-solutions ("nogoods") – for example, contradictory logical statements in a SAT problem. By avoiding these parts of the solution space one can significantly reduce the complexity of a problem. In short, a good structured search algorithm will systematically build onto goods while discarding nogoods. For example, take the vertex-coloring problem on m vertices with b colors, with e constraints involving $k = 2$ variables each (e.g., the edges): thus one immediately starts off with $z = eb$ nogood assignments of the individual constraints. Empirically, m, b, k, and e determine the difficulty of the CSP (Cer98), which can be written in terms of a critical parameter $\beta = z/m$, the average number of constraints per variable. A phase transition occurs at $\beta_c \sim b^k \log(b)$, as noted by several people (Hog) (Kirk).

Small-sized problems and large-sized problems are easy, but when the number of constraints per variable reaches this phase-transition value, the problem becomes very hard (on average). This is because heuristic searches use local information to forecast global behavior: for large values of β (tight coupling), the local information is sufficient to completely define the global behavior (making the problem easy), and for small values of β (loose coupling), statistics implies some degree of homogeneity over the problem, so that the local information is representative of the entire problem. The phase-transition paradigm for heuristic search algorithms has been much pondered upon by people interested in statistical mechanics and in computation theory.

We have now analyzed time complexity in some detail. It is possible to apply the same categorization methodology to space requirements as well. (We might care about this much more for our quantum computer, where we struggle to eke qubits out of NMR samples.) For a TM, we say that if there is a maximum number of cells $f(n)$ that it scans upon executing with input n, the language decided by the TM is in class SPACE($f(n)$); PSPACE is the class of languages decidable in polynomial space on a TM. For an NTM, $f(n)$ is defined to be the maximum over all branches which the computation might take. Many problems which are near-intractable in time have surprisingly modest space requirements, because a unit of space is a renewable resource, whereas a unit of time can of course only be used once. For example, Savitch's theorem says that an NTM which runs in $f(n)$ space can be simulated by a TM which runs in $f(n)^2$ space (the proof is by an intermediate problem called the yieldability problem), rather than the exponential requirement that we observed when analyzing time complexity. Thus NPSPACE = PSPACE, a useful result, and both P and NP are in PSPACE.

It helps also to quantify the amount of 'scratch space' needed to execute a program: L (NL) is the class of languages that are decidable on a TM (NTM) using *logarithmic scratch space*. This is a useful and very natural criterion, since a number n can be represented using $\log(n)$ bits, and it usually happens to be invariant under model/coding strategy changes (just as polynomial time is invariant under most TM model changes in the time domain). Savitch's theorem immediately gives us the fact that $\mathrm{NL} \subset \mathrm{PSPACE}$. And just as P = NP has not been settled, L = NL has not been settled. However, one can find NL-complete languages, just as one can find NP-complete languages: a function $f(w)$ is defined to be *log-space computable* if it is computable by a Turing machine which needs $O(\log(n))$ scratch space (and as much read-only and write-only tape as it needs, although it must terminate with $f(w)$ on the write-only tape, given w on the read-only tape). We say $A \leq_{\mathrm{L}} B$ if A is reducible to B, using a log space computable function f, and *NL-completeness* is defined in the obvious way. By this definition, if one can find an NL-complete language in L, L = NL, in exact analogy with the time-domain problem. An example of an NL-complete problem is the problem of determining whether there exists a directed path between two points of a directed graph (to see that this is NL-complete, for any $A \in \mathrm{NL}$ and any input w, construct a graph that represents the computation of a log-space NTM for $\langle A, w \rangle$, so that the machine accepts w if there is a path from the 'start' configuration to the 'accept' configuration).

It is easy to see that if a Turing machine M is run on input w of length n, and M runs in $f(n)$ space, and has c states and g tape symbols, then by a counting argument, the total number of configurations of M (given w) is $cnf(n)g^{f(n)} \sim n2^{O(f(n))}$ – thus, for example, if $f(n) \geq \ln(n)$, then the time complexity of a machine is at most exponential in its space complexity. This is a useful and intuitive rule: $\mathrm{NL} \subseteq \mathrm{P}$ follows naturally from it. What's interesting is that NL = coNL – the proof

involves showing that the converse of the path problem, that is, the problem of accepting graphs for which there is *no path* between two states, gives a language in NL as well (the proof proceeds by taking full advantage of the nondeterministic nature of the NTM, and counting the number of nodes that one encounters at various stages in the enumeration).

Finally, some problems are so difficult that they are considered intractable, i.e. exponential-time problems. We need more mathematics to deal with these:

A function $f(n) > \ln(n)$ is *space constructible* (SC) if the function sending the string of n 1's to the binary representation of $f(n)$ is computable, using space $O(f(n))$. This tricky definition essentially states that a program whose space requirements are bounded by an SC function can calculate the amount of scratch space that it requires, *within its own scratch space*. Thus such a function indicates a meaningful measure of the useful resources available to a program. In fact, for any SC function $f(n)$, there exists a language A that is decidable in $O(f(n))$ space but not $o(f(n))$ space (to prove this claim, create an algorithm B to decide a contrived language A by a diagonalization argument: B accepts a Turing machine $\langle M \rangle$ iff $\langle M,M \rangle$ halts *and rejects* within space $f(n)$, and so B is able to make its output different from M for all M that run in $o(f(n))$ space; there are a few tricky details dealing with the asymptotic nature of the o/O bounds (which don't apply for small n), which we do not discuss here). This is called the *hierarchy theorem*. We can prove some simple theorems, using this very important fact:

- $\forall f, g[(f(n) \text{ is } o(g(n)), g(n) \text{ is space constructible}) \rightarrow (SPACE(f(n)) \subset SPACE(g(n)))]$
- $(0 \le a < b) \rightarrow SPACE(n^a) \subset SPACE(n^b)$
- $\text{PSPACE} \subset \text{EXPSPACE}$.

Similarly, we can define $f(n) > n \log(n)$ to be *time-constructible* (TC) whenever the function mapping n 1's to the binary representation of $f(n)$ is computable in time $O(t(n))$, and we can prove that for any TC function $f(n)$, there exists a language A decidable in $O(f(n))$ time but not $o(f(n)/\log(f(n)))$. The proof is similar, with an algorithm B running M on itself, and accepting only those machines which halt *and reject* in $o(f(n)/\log(f(n)))$time; in the process of this simulation we get a $\log(f(n))$ factor due to the overhead of the process. The above 3 theorems that resulted from space considerations generalize directly to the time case. This implies that there do exist intractable problems, e.g. which require exponential resources. In fact, we can find EXP{resource}-complete problems (Sipnote2).

We can modify a recursion-theory construct, namely that of Turing-reducibility, to give the concept of relativization: we define an oracle Turing machine M^A to be one which, in an atomic instruction, can consult an *oracle for A* to decide whether a given element is in the language A. Now, since the SAT problem is NP-complete, NP and coNP are subsets of P^{SAT}. However, there exist oracles A, B such that $P^A \neq NP^A$ and $P^B = NP^B$! For example, let B be any PSPACE-complete problem, and construct oracle A so that all polynomial time oracle machines fail to decide some language L_A (e.g., enumerate A in stages, choosing oracle answers appropriately). This implies that there is something deeper going on than a mere diagonal argument or simulation argument (which is a syntactic simulation of one Turing machine by another; e.g., since we can find an oracle for B such that $P^B = NP^B$, the diagonalization argument, which is a purely formal syntactic argument, cannot separate P from NP). A new tool is needed, since these two classes are in the same 'degree.'

For quantum computation, it is useful to consider a few additional classes. The class *BPP*, which stands for 'bounded-error probabilistic polynomial' time, includes all languages which can be recognized in polynomial time with probability at least 2/3. Since iterations can reduce such error to arbitrarily small amounts, this class is generally regarded to be the class of algorithms which are "feasible in practice" (Jos97).

In classical physics, complexity is a *universal* property – it doesn't depend on the physical system running the algorithm, as far as we know. But this may just be a limitation of our imaginations, and not an inherent property of our universe.

I.1.v. Algorithmic Complexity Theory

Consider the shortest description of a number. *e*, for example, can be described by the infinite string 2.71828182845… but it can also be described to arbitrary precision by the short program

```
while(i++<N)
 a=a+1/i!;
```

The *minimal description* $d(x)$ of a string x is the shortest string $\langle M,w \rangle$ such that TM M halts with x on its tape, given input w. The length of $d(x)$, often denoted as $K(x)$, is the *Kolmogorov complexity* of x (Cov91). Clearly $K(x) \leq |x|$; it's also clear that $K(xx) \leq K(x) + c$ (where the c extra bits account for the overhead of the program that prints x twice). One can explicitly construct a TM such that $K(xy) \leq K(x) + K(y) + c$, as expected. Turing-completeness implies that $K(x) \leq K_p(x) + c$, where the p indicates another description language, and c bits are required to simulate one Turing machine on the other. An *incompressible string* is one for which there doesn't exist any c such that $K(x) \leq |x| - c$. In general, 'random' numbers are incompressible, although sometimes one can be clever and come up with a shortcut that prints out a number with little effort. For example, 26433832795028841971693993751O…11111 looks rather random, but I can write down a simple program that simply states,

print out the digits of π, starting from the 22^{nd} digit, until you reach a string of five 1's

and suddenly the description length can be seen to be much shorter. Unfortunately $K(x)$ is an uncomputable function; who knows if it's possible to create a shorter program, without simulating all possible programs? A certain program may be very short, and take an unpredictably long time to calculate the value of a certain x, but we must check it to see if it can generates x. And $K(x)$ is also a heavily language-dependent number, making its practical use almost nil. Philosophers seem to like it, though; it ties together the most abstruse parts of computation theory and information theory in an amusing formalism.

Chaitin (Cha96) has implemented axiomatic systems in LISP using Mathematica; for what it's worth, he has calculated some of the constants c that appear in the above paragraph, for a particular implementation of a Turing machine.

I.1.vi Dynamics and computation

Some dynamical systems are so complex as to be undecidable, or algorithmically random, or Turing complete, including CAs, PDEs, and neural networks (Moo90) (Omo84) (Min67). (Is there any real system more powerful than a Turing machine in that a TM cannot simulate it? Not

that we know of, due to anthropic limitations.) Moore showed that in very low dimensionality systems, e.g., a single particle in a 3D well, it is possible to have Turing-universality. The *shift map*, for example, which moves digits of elements of the 2D Cantor set (represented as a 2-sided infinite sequence of 0's and 1's, e.g. a binary number) one element to the right, is equivalent to Smale horseshoe doubling – i.e., there is an exponential increase in differences between nearby points with time, and the initial information needed to account for the system behavior for a time t is linear in t; this is a characterization of the *unpredictability* of the system (but not really the *complexity*, because to know the i^{th} digit at time t, one just has to calculate the $(i+t)^{\text{th}}$ digit).

A more complicated map, which depends on a parameter a, replaces a finite number of cells on a linear tape with a sequence $G(a)$, then shifts the whole sequence by $F(a)$, which is represented by the notation $a \to \sigma^{F(a)}(a \otimes G(a))$ (Moo90). Any such '*generalized shift*' can be thought of as a piecewise linear function on the plane, with the tape corresponding to the binary representation of a coordinate, and shifting equivalent to scaling. It squeezes and rearranges the regions where F, G are defined. This transform is smooth (discontinuities between blocks can be smoothed, and the generalized shift action is completely local: this makes it differentiable – unlike CAs which are continuous but nowhere differentiable). Classifying the behavior of different F's is very difficult, but computational universality is an interesting criterion: questions about the set of sequences on which a Turing machine will halt are questions about a *basin of attraction*.

We can replace the parameter a by the state of the head of a Turing machine. In fact, the Turing machine head state a can be absorbed into the tape t_i by placing a right after the decimal point in the binary representation of the tape, resulting in a tape $\ldots t_{-2}t_{-1}.at_0t_1\ldots$ – so if F and G read, modify, and shift the tape contents appropriately, then the generalized shift map is trivially a Turing machine. Thus in the recursion theory sense, most sets which characterize this system, like the points which are periodic under the map, or the basin of attraction, are not recursive. (Equivalently, in the computation theory sense, the languages of points which are periodic, or in the basin of attraction, are not decidable.) Therefore one cannot even decide whether or not the system is chaotic – it is much worse than chaotic; indeed, "At every time scale, new behavior shows up." One simply must wait and see what happens.

One map which has interesting dynamics, and which we will discuss in the context of quantum computing later, is the Baker's map (Arn68), which maps the unit square $[0, 1] \times [0, 1]$ to itself as follows: represent a point (x, y) in binary as $(.s_1s_2s_3\ldots, .s_0s_{-1}s_{-2}\ldots)$ and then write the two coordinates together as a string $\ldots s_{-2}s_{-1}s_0.s_1s_2s_3\ldots$; the Baker's map corresponds to shifting the decimal point one digit to the right. This corresponds to doubling the x dimension, halving the y dimension, and placing the right hand side above the left one. The name 'Baker's map' is due to the apparent similarity of these operations to a baker flattening dough, then folding it over for another round of kneading. One nice property of the Baker map is that the Kolmogorov-Sinai entropy (the incremental amount of information about the initial conditions needed to predict the behavior of a system to a certain number of iterations, given a desired accuracy) is exactly one bit per iteration.

One can also delve into cellular automata (CAs), those eerie discrete beasts that mainpulate grids full of locally interacting squares. At this point, we will not do more than mention two facts: 1) attempts have been made to classify and predict the behavior of CAs, arriving at four qualitative

classes, and 2) CAs are useful for certain differential equation simulations, in particular hydrodynamic flow simulations (lattice gases). Later we will mention CAs in the context of quantum computation.

I.2. On Quantum Computation

I.2.i. Fundamentals

A major theme of the last few decades is that information is physical. Various proposals for exotic computing devices, from vast pools of DNA to nanometer scale cellular automata to cups of coffee, have seized the popular imagination by suggesting the preponderance of incredibly powerful computing devices everywhere in nature. Unlike other schemes which gain their power through sheer volume of parallel processors or exponential quantities of energy, however, the parallel power of a quantum computer is due to its most fundamental physical properties: to completely describe a quantum system of n spins requires on the order of 2^n real numbers, which can be addressed and manipulated in various ways. But quantum information has the unique property that it is contextual, due to the strange fact that noncommuting operators cannot yield simultaneous observables (with perfect accuracy). This means that it is impossible to extract the entire exponential amount of information encoded in the state of a quantum system: only a polynomial amount of information can be read out, since a measurement collapses the exponential enormity of the state into a handful of classical values. Even so, quantum computers can do certain things better than any known purely classical computational scheme.

The fundamental element of quantum information theory is the *qubit*, which can be realized as a pair of complex numbers describing a coherent superposition of two orthogonal states which are usually labeled 0 and 1. Just as a bit is a fundamental abstraction which gave rise to communication theory, computer science, and information-theoretic models of almost every discipline ever created, the qubit is the fundamental abstraction of quantum information theory. Observing a single qubit without any prior knowledge yields only one bit of information, since the qubit collapses into either a classical 0 or 1 upon observation. Thus a qubit has the form

$$|a,b\rangle = a|0\rangle + b|1\rangle$$

where a and b are complex numbers satisfying $|a|^2 + |b|^2 = 1$; it takes 3 real numbers to specify a qubit entirely. A qubit can be thought of as a single field mode, a spin, a suitably processed state of a thermal ensemble of quantum particles, or any other suitable system with two distinct levels for which superposition is not precluded by any superselection rules. Any computation, or operation on a set of qubits must be unitary and linear, for otherwise it cannot represent coherent evolution under a Hamiltonian – i.e., it would be impossible to implement without destroying the coherence which enables an isolated quantum system to be meaningful. Therefore quantum computation is of necessity a reversible process, which means that classical computations cannot be implemented as-is. (We discuss the process of making algorithms reversible at a later point.)

Quantum measurement of commuting observables in any set of basis states must be carried out at some point, because our consciousness is classical (why it is so, we are not quite certain, but look at (Eve73) for one well-popularized hypothesis). For bulk NMR quantum computing the observable is the transverse magnetization of a set of spins as it precesses in a magnetic field; we

will later discuss how to extract the maximum amount of classical information from an NMR signal.

This conventional paradigm, where a classical computer sends commands to a quantum computer in order to keep it under tight control, is called the *quantum coprocessor* paradigm. A free-running quantum computer, while interesting and unlimited by classical decidability restraints (see the next section), is unfortunately incomprehensible.

The nature of quantum parallelism is worth pondering: it is not merely due to interference effects (often called *coherence* or *superposition* in classical mechanics and classical electrodynamics). One can consider, as a quantum computing model, a cascade of branching processes, for example the paths taken by photons through a set of parallel diffraction gratings. A photon's path can be considered to be equivalent to the execution of a program; thus n^k programs can be coded by, say, an array of k gratings with n slots each, perhaps with different phase shifters or other devices attached to them. Then n^{-k} is the probability that a particular program of length k is executed, and we see that each signal bears a concomitant exponential loss in intensity. Thus there is a hidden exponential resource here, namely the amount of energy required to see a constant-intensity result at the end of any particular photon's path. So superposition of waves alone is not enough to overcome classical exponential limits; if so, water waves in a tank, or modeshapes on a string, would suffice!

For example, Cerny (Cer93) proposed a solution for the traveling salesman problem using a model of quantum computation similar to the above: consider a TSP on n cities, and a series of $(n\text{-}1)$ diffraction gratings, each with n-1 slits, for a total of $(n\text{-}1)^{n-1}$ possible trajectories. Every quantum has a finite probability of passing through all the trajectories in polynomial time – but in order to get at least one quantum in each path, one must have $O((n\text{-}1)^{n-1})$ photons, an exponential quantity of energy.

The fundamental observation (Jos97) is that *entanglement*, not superposition, is the essential feature that empowers quantum computation, and is what gives other quantum technologies (such as quantum teleportation) their power. That is why the simple act of sending photons through slits, described in the last paragraph, falls prey to the exponential clause of the strong Church-Turing thesis: superposition must occur not just in space or in time, but in the *exponential-dimensional tensor product space of states of a quantum system*. Simple superposition without the mutual inseparablity of entanglement result in rather boring Cartesian product spaces, with the equally boring property of lacking computational power. Ironically, since entanglement is so delicate and easily upset by parasitic interactions with the environment, this also means that quantum computation is a very delicate thing to implement. On one hand, the quantum computer programmer must be able to address individual qubits and manipulate them; on the other hand, this hardware must not measure or otherwise disturb the system when manipulations are not desired.

We now examine this mysterious phenomenon, entanglement, at some length.

I.2.ii. Quantum Mechanics and Information

Quantum-mechanical thermodynamics *is* the heart of thermodynamics, accounting for such phenomena as spin-statistics, correct Boltzmann counting, and most of the interesting many-body system characteristics such as superfluidity and Fermi pressure. At its most basic level, to put this discipline in an information-theoretic context requires generalizing the idea of information.

Because of the intricate correlations that quantum systems can possess, they are difficult to interpret within the bounds of classical information theory. But a complete theory which accounts for both the quantum and classical cases has been developed (Cer97b). The von Neumann entropy for a quantum system, represented by a density matrix ρ_A, where A is a quantum system or quantum source,

$$H(A) = -Tr\rho_A \log \rho_A$$

is familiar to students of statistical mechanics for being a measure of the disorder of a quantum system (see the explanation in **I.1.i.**), but it was recently established that $H(A)$ is the minimum number of qubits needed to losslessly code an ensemble of quantum states, thus giving information-theoretic importance to this representation – see the argument later in this section for a summary of the proof (Sch95). Further, if ρ_A is composed of orthogonal quantum states it clearly reduces to a classical Shannon entropy $H(A)$, since we can just diagonalize ρ_A in the orthogonal basis; we will discuss this fact later in the context of quantum channels. For two operators A, B, the definitions of joint, conditional, and mutual entropy can be defined as follows, where ρ_{AB} is the joint density matrix:

$$H(A,B) = -Tr\{\rho_{AB} \log \rho_{AB}\}$$
$$H(A \mid B) = -Tr\{\rho_{AB} \log \rho_{A|B}\}$$
$$I(A,B) = Tr\{\rho_{AB} \log \rho_{A;B}\}$$

The definitions of $\rho_{A|B}$ and $\rho_{A;B}$ are surprisingly subtle. They are given by

$$\rho_{A|B} = 2^{-\sigma_{AB}},$$
$$\rho_{A;B} = \lim_{n\to\infty}\left[(\rho_A \otimes \rho_B)^{1/n}\, \rho_{AB}^{-1/n}\right]^n$$

respectively, where

$$\sigma_{AB} = 1_A \otimes \log_2\rho_{AB} - \log_2\rho_{AB}$$

and

$$\rho_A = \mathrm{Tr}_B\{\rho_{AB}\},\ \rho_B = \mathrm{Tr}_A\{\rho_{AB}\}.$$

(As noted earlier, Tr_B means to take the partial trace of the density matrix; e.g.,

$$\langle a|\rho_A|a'\rangle = \sum_b \langle ab|\rho_{AB}|a'b\rangle.$$

This is where the duality between projection operators and density matrices becomes useful.) The justification of these choices is basically *by analogy*: as ρ_{AB} becomes diagonal, for example, the quantum conditional entropy converges to its classical value; also, $\rho_{A;B}$ satisfies the classical identity $I(A,B) = H(A)+H(B) - H(A,B)$. But $\rho_{A|B}$ and $\rho_{A;B}$ are not true density matrices. For example, $\rho_{A|B}$ may have eigenvalues greater than 1, when σ_{AB} has negative eigenvalues – in such situations one obtains negative conditional entropies! It has been shown that a separable density matrix ρ_{AB} (i.e., one that can be expanded in the form

$$\sum_i a_i {\rho_A}^i \otimes {\rho_B}^i$$

and therefore can't be told apart from a classical ensemble containing two completely decoupled systems) must have positive semidefinite σ_{AB} and thus positive conditional entropies (Cer97a), but sufficiently entangled systems can indeed guarantee strange behavior like negative conditional entropies. And oddly enough, $I(A, B)$ can exceed the entropy of each individual set of variables! The above definitions and generalizations thereof have been applied to quantum teleportation and superdense coding (i.e., where two classical bits are encoded in one qubit) in the paper (Cer97b). In this paper the authors treat entangled qubits as virtual conjugate particles e, $\bar{e}$ (collectively called *ebits*, or entangled qubits), carrying opposite information content (± 1 qubit each), and suggest that $\bar{e}$ may be visualized as e going backwards through time! One can, for example, draw a Feynman diagram for ultradense coding (consumption of a shared entangled state, plus transmission of one qubit, is equivalent to transmission of two classical bits). Thus these 'information quanta' suddenly have many properties analogous to those of virtual particles in relativistic quantum field theory – in particular, only through interactions that result in classical bits may they become visible to the world. Importantly, this formalism allows an intuitive picture of conserved information flows. *Ebits* can be created simply by transmitting one bit of a Bell pair while keeping the other, but one cannot create a qubit directly from an ebit (some classical bits are required). An interesting point is that if the ebit used for teleportation or superdense coding is only partly entangled, then the transmission will appear noisy.

The *Schmidt decomposition*, sometimes called the *polar decomposition*, is a way of representing an entangled *pure* state in the form

$$\psi = \sum_{i=1}^{d} c_i \left| a_i \right\rangle \otimes \left| b_i \right\rangle$$

where the c_i are positive real coefficients, and $\left| a_i \right\rangle$, $\left| b_i \right\rangle$ are appropriately chosen orthonormal states of two subsystems A, B, which may perhaps be located far apart from one another (Ben95b) (Sch95). It is useful to note that local operations cannot increase the number of nonzero terms in the Schmidt decomposition; i.e., one cannot create entanglement with purely local operations (although by consuming other entangled bits, one can increase the entanglement of another subsystem). Local unitary operations on subsystem A can only change the eigenvectors of A (not the eigenvalues), and cannot affect B's observations at all. As noted above, in this basis σ_{AB} is positive semidefinite. If A makes an observation of his part of the system, this is equivalent to him first tracing over the degrees of freedom of B's subsystem (which results in an apparent mixed state of his subsystem, represented by a diagonal density matrix ρ_A in the Schmidt basis), then making an observation on this reduced density matrix:

$$\rho_A = Tr_B \left| \psi \right\rangle \left\langle \psi \right| = \sum_{i=1}^{d} {c_i}^2 \left| a_i \right\rangle \left\langle a_i \right|$$

Then the *entropy of entanglement* is defined to be

$$E = -Tr \rho_A \log(\rho_A) = -Tr \rho_B \log(\rho_B) = -\sum_{i=1}^{d} {c_i}^2 \log({c_i}^2) .$$

Note that it is just the Shannon entropy of the squared density matrix coefficients in the Schmidt basis. $E = 0$ for a direct product, and $E = 1$ for a Bell state. It can be proven (Pop97) that this is the unique measure of entanglement for pure states. The argument, like Shannon's definition of

information, is based on two axioms: it is impossible to increase entanglement purely by local operations, and it is possible to reversibly transfer entanglement from some set of shared ebits to previously unentangled qubits (leaving the original ebits unentangled) (Ben96c). (The last statement is true asymptotically; one can reversibly transform k systems in one entangled state into n systems in pure singlet states only in the limit $n,k \to \infty$; in fact n/k is often asymptotically an
irrational number, so due to incommensurability such reversible procedures can't even exist except in the infinite limit!) Formally these two statements are akin to the Second Law of thermodynamics and the statement that all reversible heat engines are equally efficient, so the derivation of the entropy of entanglement is completely analogous to the derivation of entropy in thermodynamics! Thus, proceeding as in thermodynamics, we require the degree of entanglement to be extensive, we can measure the entanglement of an arbitrary state by transforming it into a set of singlet states (which are defined to have entanglement equal to 1), and so on. Unfortunately this definition is only good in the quantum analogy to the 'thermodynamic limit,' meaning access to infinitely many pure ebits, which rarely occurs in real life. Also, nobody has come up with a measure of entanglement for a mixed state, since no on has exhibited a reversible way to convert a density matrix into pure states; candidates include the number of singlets that can be packed into a density matrix, or the number which can be extracted – sometimes these two values can be different (Ben96d). Therefore many other definitions may be more practical, if appropriately justified; it is difficult to imagine what physical manifestation this measure of mixed-state entanglement might take. Indeed, the recent literature indicates that even respectable scientists occasionally need reprimanding due to the ambiguity of dealing with issues as nonintuitive as entanglement (Unr).

One interesting manifestation of these results is that two people with many partially-entangled qubits can increase the degree of entanglement of a few qubits to arbitrary purity, at the expense of the others. Note that this is not in contradiction of our above statements, since only the *total* entanglement is required to be conserved. For a large number n of entangled pairs of two-state particles, each with entropy of entanglement $E < 1$, the yield of pure singlets goes like nE–O(log(n)) (compare this to the pure state yield of bulk spin resonance quantum computation, below). The process of *Schmidt projection* is as follows: suppose that the initial system is in the product state

$$\psi = \prod_{i=1}^{n} (\cos\theta \left| a_{1i} b_{1i} \right\rangle + \sin\theta \left| a_{2i} b_{2i} \right\rangle)$$

which has 2^n terms, each with one of $n + 1$ distinct coefficients $\cos(\theta)^{n-k}\sin(\theta)^k$. We can treat these states as $n + 1$ orthogonal subspaces, labed by the power k of $\sin(\theta)$ in the coefficient. A and B project the state ψ onto one of these subspaces by making an observation, and obtaining some k; this collapses the state down into a *maximally entangled state*, which occupies a $2n!/(n-k)!k!$ dimensional subspace of the original space; if lucky, one can get even more ebits than one started with (but the *expected* entanglement cannot be greater than before). It's analogous to the method of taking a biased coin, say with probability of heads .3, and getting an unbiased random bit by flipping the coin and keeping sequences HT = 1 and TH = 0 (probability .21 each), and discarding TT and HH (probability .49 and .09, respectively). In our case, measuring a power k symmetrizes our state by selecting all possible combinations with some equal probability (although we cannot actively choose that probability) and discarding all the states with

probabilities that differ from it. One can transform this new state formally into singlets as follows: measure k for each of m batches, each containing n entangled pairs, getting k_i, $i = 1..m$. Let D_m be the product of all the coefficients $n!/k_i!(n-k_i)!$, $i = 1..m$, and continue until $D_m \in [2^l, 2^l(1+\varepsilon)]$, where ε is the desired maximum error in the entanglement, and l is some integer. Then A and B make a measurement on the system which projects it into one of two subspaces, one of dimension 2^{l+1} (which is in a maximally entangled state of two 2^l-dimensional subsystems) and one of dimension $2(D_m - 2^l)$ (which discards all the entanglement). In the former case, we have again effectively symmetrized the state; in the latter, we lose everything. (Such is the risky nature of quantum engineering.) Finally, another round of Schmidt decomposition arranges the density matrix into a product of l singlets.

Quantum channels, over which qubit messages are sent, are a useful abstraction. In classical information theory, a source A produces messages a_i with probability $p(a_i)$, and the *fidelity* of the coding system is the probability that the decoded message is the same as the transmitted one; in quantum information theory a *quantum source* codes each messages a from a source A into a *signal state* $|a_M\rangle$ of a system M; the ensemble of messages is then representable by a density matrix

$$\rho = \sum_a p(a)\pi_a \text{, where } \pi_a = |a_M\rangle\langle a_M|,$$

and the *fidelity* for a channel where r_a is received when π_a is transmitted is defined to be

$$f = \sum_a p(a)\mathrm{Tr}(\pi_a r_a)$$

(so that for perfect noiseless channels $f = 1$) (Sch95). Compare this to our definition of classical probability fidelity in **I.1.ii.**; we will unite these pictures in the next paragraph. Another possible definition of a quantum channel W, which lends itself more to problems involving noise, is to consider a channel to be a probability distribution on unitary transformations U which act upon $H_{signal} \otimes H_{environment}$; before reading a quantum channel, one must take a partial trace over $H_{environment}$, and one can define the fidelity as

$$\min_{\text{all signals x}} \langle x|W|x\rangle$$

(Cal96). Note that there are subtleties for the quantum channel which one does not have to consider in the classical case: for example, one cannot copy an arbitrary quantum signal (the "no-cloning" theorem (Woo82)); the proof follows immediately from the linearity of quantum mechanics. Only if the signals are orthogonal can they be copied (i.e., measuring a system which is known to be in an eigenstate is trivial, and yields all the information available about the system). Fundamental to quantum communication is the idea of *transposition*, or placing a system Y in the same state as a system X (perhaps resetting X to a useless state in the process); unitary tranposition from system X to system Y can occur iff the states in X have the same inner products as the corresponding states in Y, e.g. $\langle ax|bx\rangle = \langle ay|by\rangle$, which occurs iff the Hilbert space of Y is of dimension no less than the Hilbert space of X, for obvious reasons. If a message is unitarily transposed from a source A to a system M, which upon transmission is decoded into a message A' (on an isomorphic quantum system) by the inverse of the unitary coding operator,

$$A \to M \to A'$$

then we say M is the *quantum channel* for the communication. The question arises, when is the quantum channel M big enough for a particular quantum source A?

The von Neumann entropy of an ensemble of quantum messages ρ equals the Shannon entropy of the source *A* if the $|a_M\rangle$ are orthogonal, else $H(\rho) \leq H(A)$. We define a *positive-operator valued measure* (POVM) to be a set of operators $\{ O_i \}$ with exclusively positive eigenvalues which are complete (e.g., their sum is the identity operator). We can cast the quantum fidelity between two specific states ρ_0 and ρ_1 as

$$F(\rho_0, \rho_1) = \min_{\{O_i\}} \sum_i \sqrt{(\mathrm{Tr}(\rho_0 O_i))(\mathrm{Tr}(\rho_1 O_i))} \,,$$

which considers all possible interpretations of two specific states (minimizing over the set of all possible measurements), as opposed to a specific interpretation applied to all possible states (as in the paragraph above, where we minimized with respect to all possible signals). Although this new picture is not suitable for evaluating signals traveling through a quantum channel, where we want to consider many signals and a single measurement, this version of the quantum fidelity does have a very simple closed form, as shown in (Fuc96):

$$F(\rho_0, \rho_1) = \mathrm{Tr} \sqrt{\rho_1^{1/2} \rho_0 \rho_1^{1/2}}$$

which, in the case that ρ_0 and ρ_1 correspond to pure states, reduces just to the inner product between the pure states (Woo). Fuchs' simple proof of this last formula applies the Cauchy-Schwarz inequality to the sum over the elements of the POVM, then applies the completeness criterion for POVMs to eliminate the arbitrary set $\{O_i\}$ from the equation entirely.

We briefly mention a few more inequalities, without proof, for the interested reader. (Most of the proofs use variants of classical inequalities for sums and vector spaces, like Holder's inequality, Jensen's inequality, or Minkowski's inequality.) *Kholevo's theorem* (Kho73) bounds the information one can get about a quantum channel *M* by measuring it in the basis of messages of a quantum source *A*: suppose the output of the quantum channel is described by a density matrix

$$\rho = \sum_m p(m) \rho_m$$

where we label the different message states *m* with individual density matrices ρ_m, to indicate that they might not be pure states. Then, for any measurement strategy *A*, the mutual information obtained from measurements on *A* and on *M* satisfies

$$I(A, M) \leq H(\rho) - \sum_m p(m) H(\rho_m) \,,$$

where $H(\rho_m)$ is the entropy of the density matrix for the m^{th} signal state (if the signals are encoded in pure states, then this entropy is zero) (Sch90, in Zur89). The proof is given in some exquisite detail in (Fuc96).

This can be used to prove, for example, a conjecture by Everett (Eve73) that for systems ρ_1, ρ_2 with respective observables A_1, A_2,

$$I(A_1, A_2) \leq S(\rho_1) \text{ and } I(A_1, A_2) \leq S(\rho_2) \,.$$

It might be interesting to see if these results could be proved more easily using the formalism in (Cer97b) outlined above.

Another inequality, an information-theoretic version of the Heisenberg uncertainty principle, was discovered by Maasen and Uffink (Maa88) (Sch90), and states that for observables *A*, *B* with eigenstates a_i and b_i and a state (or message) ρ,

$$H(A \mid \rho) + H(B \mid \rho) \geq -\log(\sup_{i,j} |\left\langle a_i \middle| b_j \right\rangle|^2).$$

As explained in (Sch90), since $H(A \mid X) = H(A) - I(X, A)$, for a message X embedded in an N-dimensional Hilbert space, this can be written as

$$I(X, A) + I(X, B) = H(A) + H(B) - (H(A \mid X) + H(B \mid X)) \leq 2\log N + \log(\sup_{i,j} |\left\langle a_i \middle| b_j \right\rangle|^2).$$

One can derive a *noiseless coding theorem* for quantum information theory, analogous to the theorem of the same name in classical information theory (see **I.1.ii.**), which states that as the length of a string of messages from a source goes to infinity, the mean number of bits per message can be brought arbitrarily close to the Shannon entropy of the source, but no less. Similarly, for the quantum case, the number of qubits per message can be brought arbitrarily close to the von Neumann entropy of the signal source (Sch95). We model the process as follows: a source *A* encodes its message *a* in a state $\left| a_M \right\rangle$ of a system *M* (which, as an ensemble, we can therefore describe as

$$\rho = \sum p(a)\pi_a ,$$

in density matrix form) which is then transposed through a channel *X*, then transposed to a receiver state *M'*. We model the channel *X* with two components, *C* (channel) and *D* (discarded). Discarding a component *D* can be simulated by taking the *partial trace* Tr_D of the system, so that all information about *D* is lost. Thus the process looks like

$$M \rightarrow C{+}D \rightarrow C \rightarrow C{+}D' \rightarrow M'$$

where in the next-to-last step, we must make up some numbers for the state of *D'* so that the decoding algorithm will work properly; adjoining a quantum system in this fashion is done by taking the *tensor product* of the original system with the adjoined system. The message therefore undergoes the following transformations as it passes through the above communications channel:

$$\pi_a = \left| a_M \right\rangle\left\langle a_M \right| \rightarrow U\pi_a U^{-1} \rightarrow Tr_D U\pi_a U^{-1} \rightarrow Tr_D U\pi_a U^{-1} \otimes D' \rightarrow r_a \equiv U^{-1}(Tr_D U\pi_a U^{-1} \otimes D')U$$

The fidelity, as noted earlier, is equal to

$$\sum_a p(a)\,\mathrm{Tr}(\pi_a r_a),$$

and gives the probability that the received message matches the transmitted one (under any possible measurement). Two facts about channels that link dimensionality to channel capacity are useful: (1) If *C* is *d*-dimensional, and for any projection *P* onto a *d*-dimensional subspace of *M*, $\mathrm{Tr}(\rho P) < \eta$ (or, alternately, that the sum of any *d* eigenvalues of ρ is less than η), then the fidelity $f < \eta$. (Proof: when we arbitarily adjoin a pure state *D'* to the message from *C* and decode to form r_a, the useful information is to be found only on a *d*-dimensional subspace of *M'*,

$$r_a = \sum_{i=1}^{d} b_i \left|\phi_i\right\rangle\left\langle\phi_i\right|;$$

evaluating the expression for *f* with this expression and recalling $b_i < 1$ in conjunction with the hypothesis of the lemma, gives the desired answer.) (2) If, in the above scenario, $\mathrm{Tr}\,\rho P > 1-\eta$ for some projection *P*, then there exists some communications scheme with $f > 1-2\eta$. (Proof outlined in (Sch95note).)

The *quantum noiseless coding theorem* states that if $H(\rho)+\delta$ qubits are available per M signal then sufficiently long signal sequences can be transposed using these qubits with fidelity greater than $1-\varepsilon$ for all ε between 0 and 1; conversely if $H(\rho)-\delta$ qubits are available, for sufficiently long sequences the fidelity will be less than ε for all ε between 0 and 1. Consider a channel C consisting of k qubits and a message source M^N consisting of N copies of some D-dimensional system M described by

$$\rho = \sum_{a=1}^{D} p(a)\pi_a \ .$$

The composite system M^N can be thought of as being in a state described by some "message string" $\left|a_1 a_2 ... a_N\right\rangle$, and the statistics of M^N can be described by a density matrix ρ^N which is the tensor product of N copies of ρ. Since the eigenvalues of ρ formally satisfy a probability distribution whose Shannon entropy is equal to the von Neumann entropy of ρ, and the eigenvalues of ρ^N are just the D^N possible products of N eigenvalues of ρ, it follows that each eigenstate of ρ^N corresponds to a message string which occurs with probability equal to the corresponding eigenvalue of ρ^N. Having established the connection between Hilbert space dimension and von Neumann entropy in the two lemmas in the previous paragraph, and having seen the connection between eigenvalues and classical probabilities, the proof is now obvious, since the machinery of information theory completes the proof just as we did earlier in **I.1.ii.** Earlier we asked how the entanglement of a pure state might be affected by transmission through a channel. As noted above, entangled states and quantum message sources are formally identical, since both can be represented by a density matrix, so the above proofs and theorems carry over exactly to the entanglement case. The algorithm for performing noiseless coding is given in explicitly reversible high-level pseudocode in (Cle96), in which it is shown that n-qubit strings can be encoded in $O(n^3)$ time with $O(n^{1/2})$ scratch qubits. The primitives in **Appendix A** suffice to implement much of the Schumacher coding operations.

Finally, one might ask whether it is possible to communicate in the presence of noise, i.e. whether *quantum error correction* is feasible. It was originally believed by many people (including Landauer) that this was impossible, because one cannot perform regeneration, as one does with a digital computer by using devices with nonlinear input-output relations. Mere copying, distributed storage, or redundancy cannot be used to perform error-correction, due to the no-cloning and demolition-measurement properties of quantum mechanics. One can treat an error as an interaction with or measurement by an 'environment' system; the key insight to preventing revelation by this interaction is therefore to hide the desired bits from the environment. For example, information must be embedded entirely in the entanglement of various bits (without respect to the basis, since the environment could be "measuring" with respect to any basis); furthermore the error must occur in a subspace orthogonal to the space of the coded information (DiV97b), so that it can be observed and proper action taken.

In the past two years many proposals have been made which develop quantum error correction and refine it to a discipline of fault-tolerant quantum computation, starting with (Sho95) and (Ste96), in which one qubit is encoded in 9 qubits and 7 qubits, respectively. At least 5 qubits per encoded qubit are necessary to correct one general error (DiV97b). In its most basic form, error correction is formally the embedding of k qubits in a subspace of an n-dimensional space so that

decoherence of any *t* qubits doesn't affect the possibility of reconstructing the original *k* qubits of information (Cal96).

For example, here is Steane's 7-qubit code, which corresponds to the [7, 4, 3] Hamming code (Ste96). Consider a codeword *v* from the set of 16 codewords which are orthogonal (mod 2) to the matrix

$$H = \begin{bmatrix} & & & 1 & 1 & 1 & 1 \\ & 1 & 1 & & & 1 & 1 \\ 1 & & 1 & & 1 & & 1 \end{bmatrix}.$$

Each of these 16 codewords is a 7-dimensional vector, each coordinate of which is 0 or 1. Eight of the codewords have even Hamming weight (that is, they contain an even number of 1's) and eight have odd Hamming weight. If bit *i* of the codeword *v* flips, giving $v + e_i$, then applying *H* to the modified codeword gives He_i, which uniquely identifies the error and allows it to be corrected for, by toggling the i^{th} bit. But *H* reveals no information about which code word was being corrected. If the qubit $a|0\rangle + b|1\rangle$ is affected by an error, but we encode $|0\rangle$ as the equal superposition of the eight even-Hamming-weight codewords, and $|1\rangle$ as the superposition of the eight odd ones, then by appending an ancilla qubit (initialized to the state 0), one can apply *H* to the combined system to get

$$H: |v,0\rangle \rightarrow |v, Hv\rangle .$$

Reading the ancilla bit allows amplitude errors to be corrected. Applying the Hadamard matrix (see section **II.2.**) to rotate the phase errors into amplitude space, and repeating the application of the matrix *H*, we can correct for a phase error as well, if no amplitude error was found in the first step. If a code is designed to resist one error, however, and two happen to occur, then using the above naïve method can actually make things worse, since errors can propagate. One can reduce the probability of error to e^2, where *e* is the probability of having a single-bit error, by using two ancillae, or even more. Picking a code appropriate for the number and type of errors that are expected, is essential for error correction to be meaningful.

Finding such a meaning, however, can be very tricky. A code which has been experimentally verified (Cor98) is a simple 3-qubit 'majority code' (e.g., if two agree and the third is different, choose the message given by the two which agree) which had been modified for the quantum regime, but it only corrects some errors: in particular, small random phase fluctuations, due to local magnetic field inhomogeneity (e.g., diffusion in a *B*-field gradient).

Some vocabulary from classical coding theory is essential. For binary computers the simplest codes to consider are those over $F_2 = \{0,1\}$. A *linear code* of length *n* over F_2 is a set of binary strings of length *n*, which comprise a vector subspace of F_2^n. The *Hamming weight* $H(w)$ of a code word *w* is the number of 1's in the code word, and the *Hamming distance* between v_1 and v_2 is $H(v_1 + v_2)$, which counts the number of places where v_1 and v_2 differ. The *minimum distance* of a code is the minimum distance between two distinct codewords; for a linear code it is simply the minimum nonzero Hamming weight. A linear code is completely specified by its length *n*, dimension *k* (as a subspace of F_2^n), and minimum distance *d*; such a code is called an [*n*, *k*, *d*] code. It is clear that an [*n*, *k*, *d*] code can correct $t \equiv \lfloor (d-1)/2 \rfloor$ errors, since even if two different

codewords were each corrupted with *t* errors, they could not possibly equal the same string, and so the corruption must be invertible. The *rate* of a [*n*, *k*, *d*] code is *k*/*n*, and gives the relative fraction of each codeword that contains real information. The *dual code* $C^{\perp}$ of a code *C* is the set of vectors perpendicular to all the codewords of *C* – i.e., for any *v* in $C^{\perp}$ and *w* in *C*, the number of places where *v* and *w* are both 1 must be even. A code can be self dual, e.g. the very simple [2, 1, 2] code {11, 00}.

A *quantum error-correcting code Q* with rate *k*/*n* is a unitary map $H_k \rightarrow H_n$ of *k* qubits to a subspace of an *n*-qubit space (Cal96). We can model *t* errors by applying a random unitary transform to some space $H_t \otimes H_{env}$, $H_t \in H_n$, and then taking the partial trace of the system with respect to H_{env}. Explicitly, we can map a [*n*, *k*, *d*] linear code C_1 into a subspace of H_n, say H_1. Let C_1 be the row space of a $k \times n$ matrix *M* (the *generator* of C_1), and for each *w* in F_2^k define quantum states $|c_w\rangle$ in H_n as

$$2^{-k/2} \sum_{v \in F_2^k} (-1)^{vMw} |vM\rangle$$

Now $\{|c_w\rangle \mid w \in F_2^n / C_1^{\perp}\}$ forms a basis for H_1, as one can see by checking dimensions and linear independence. Thus, if a second *m*-dimensional linear code $C_2 \subset C_1$ exists, we may define our quantum codewords to be the codewords in $C_2^{\perp}$, which are in one-to-one correspondence with the elements of $C_1^{\perp} / C_2^{\perp}$, and which has a rate (*k*-*m*)/*n*. If C_1 and $C_2^{\perp}$ have minimum distance *d*, this code corrects $t = \lfloor (d-1)/2 \rfloor$ errors.

Why should this be so? The reasoning is that in the qubit basis, *t* amplitude errors can be corrected by classical error-correction, and in the Bell basis, where phase errors have been transformed into amplitude errors, *t* phase errors can thus corrected as well. Once again the process has been to take a classical idea, embed it in a quantum framework, and then to show that the classical intuitions hold. Of course, the proof that this works (given in (Cal96)) requires much additional logic – for example, they show that the two correcting stages do not interfere with one another, and also that this classical error-correction does not actually destroy any encoded information. We will not go through this here, since there are no essential ideas in the proof that are not contained in the above discussion on Steane's 7-qubit code.

One can use supersets of these error-correction ideas to prove the *quantum noisy coding theorem*, which like the classical noisy coding theorem of Shannon, indicates the amount of information which can be transmitted perfectly over a noisy channel, using error correction as necessary. Schumacher et. al. (Sch96) give an existence proof of this phenomenon: they show that for appropriate signal sources, there exists a superoperator which can correct the errors induced by a noisy channel.

I.2.iii. Quantum Logic

Models of quantum logic have been proposed by von Neumann and others (Neu55). By quantum logic I do not mean the system of logical gates that I will describe in the following sections, but the actual formalism associated with a quantum 'propositional calculus' (Svo94) – a link

between syntactical and semantic constructs which allows symbolic manipulation to reveal truths about quantum mechanics. This is an interesting idea, since quantum mechanics currently has no precise syntax for describing systems so that a machine might be able to parse and decide arbitrary statements (Rot97). Questions about the decidability of certain statements about quantum mechanical states may remain unanswered due to the informal treatment usually accorded the mathematical structure of quantum mechanical axioms. The basic structure of the logical system comes from formally interpreting the various properties of quantum systems within the infrastructure of an *n*-dimensional Hilbert space *H*. A contradiction is associated with a 0-dimensional subspace, and a tautology with the entire space *H*. Truth valuations are associated with projection operators P_i, and each proposition which has a binary truth valuation is associated with a linear subspace of the Hilbert space. Logical operators like intersection $\cap$, union $\cup$, and negation $\neg$ correspond to subspace intersections, subspace direct sums, and orthogonal subspaces, respectively. As formulated by von Neumann, the DeMorgan laws clearly do not apply. Kochen and Specker, who are famous for their assertion (and proof) that hidden-variable theories of quantum-mechanics must be contextual (Gil), went farther than von Neumann by trying to define incompatible propositions as those which cannot be simultaneously measured. They defined a commeasurability operator ♥, such that ♥(P_1;P_2) asserts that propositions P_1 and P_2 are simultaneously measurable. They also showed that there exist classical tautologies which are not true according to 'quantum logic,' which is not surprising given the very different axioms and rules of derivation which guide classical and 'quantum logic.' (The shortest known such tautology is the implication of the disjunction of 16 well-formed formulas in 11 letters! Each letter can be pictured as a ray in 3-dimensional space.) It is quite possible that none of this is interesting in a physical sense, since most physicists rely on their intuition about quantum mechanics to solve problems, rather than depending upon the precise derivation and enumeration of theorems about quantum mechanics from a set of axioms – so perhaps quantum logic will be useful only in a very abstract mathematical sense. Certainly the discovery of a paradox or of a theorem which applies to all models of a Hilbert space would be very useful. At least one author asserts that quantum logic is a useful tool for studying Hilbert spaces (Coh89).

"There are people who live as animals, and people who live as men, and people who try to live as gods. The romantics among us would say that it boils down to an individual's capacity for passion, but the realists know that it depends only on one's model of the universe."
– E

II. Models of Quantum Computation

II.1. Preliminary models

The most fundamental constraint of quantum algorithm design is that for a completely described system, any time-evolution operator, and therefore any meaningful quantum computation, is unitary – and therefore reversible. This statement is equivalent to the fact that for a completely described quantum mechanical system, the probabilities of all distinct experiment outcomes must sum to 1 (or, for continuous states, the correspondin probability measure must have integral 1). Before implementing a quantum algorithm, one must explicitly cast it in a reversible form. This can be done by maintaining 'scratch bits' which are then removed by undoing parts of the computation, a painstaking task (Pre96). Perhaps we should briefly examine classical reversible computation to gain some intuition for this 'reversibilization' process.

The Bennett scheme for embedding an irreversible TM in a reversible TM is very simple – almost astonishingly so, given the number of great minds which historically expressed disbelief at the possibility of performing such an embedding. Simply make a TM with three tapes, (`input`, `history`, `output`) – the standard TM tape with the input on it, an ancillary history tape, and an extra output tape (Ben82). We will perform a computation *C* using only reversible operations, but ending with the correct answer on the output tape. Begin the computation in the state (`input`, - , -), and perform the computation *C*, recording the entire history of what occurs, resulting in (`output`, `history of C`, -). Then copy the output to the extra output tape, giving (`output`, `history of C`, `output`), and then undo the operations of *C* in reverse temporal order of their execution as recorded on the history tape, resulting in (`input` , `-` , `output`). This concludes the embedding of an arbitrary computation *C* in a reversible framework.

If *C* already is a 1-1 computation, so that its inverse C^{-1} exists, then we can do even better: after completing the last step, swap the input and the output, getting (`output`, `-` , `input`). Then execute C^{-1}, getting (`input`, `history of` C^{-1}, `input`). Delete one copy of the input (this step is reversible since we have two copies of the same information), getting (`input`, `history of` C^{-1}, -), and finally execute *C* one more time to get (`output`, - , -). Thus in this case, the reversible computer can completely emulate the irreversible one – with the expenditure of a linear increase in operation time and a possibly large increase in the amount of space needed.

Almost all quantum computations which are based on implementing irreversible calculations proceed by using some variation of book-keeping and undoing. The quantum Turing machine (QTM) model has been suggested as a way of examining unitary execution of a quantum algorithm. One way to construct the QTM is to quantize the ordinary universal Turing machine, and replace the finite state machine in the TM head by a unitary (and therefore reversible) version (Svo94). A universal QTM *U* with an infinite qubit tape and with state vector ψ_U can be defined by three types of operators, corresponding to the three components of a classical Turing machine configuration: the tape position operator $\hat{x}$, the memory operator $\hat{m}_i$, which measures

the state of the i^{th} tape qubit, and the head operator $\hat{p}$, which measures the state of the head (where the Hilbert spaces of the head operator and of each tape memory operator are finite-dimensional). It follows that $|\psi_U\rangle$ is a vector in the Hilbert space of computational basis states

$$|x\rangle \otimes |p\rangle \otimes |\vec{m}\rangle,$$

where *x*, *p*, and $\vec{m}$ are the eigenstates of their respective operators. The amplitude of $\vec{m}$ in $|\psi_U\rangle$ indicates the program to be executed by the universal QTM, as well as the input, and therefore specifies a unitary operator which operates on the state of the head. Curiously, the tape position operator can be 'anywhere', but it still is limited to moving one cell to the left or to the right! Benioff (Ben97b) has proposed a cogent model of QTM dynamics in terms of the unitary *step operator T* that advances a computation by one step. *T* expresses the head-tape coupling (with appropriate locality constraints). There are many choices of *T*'s locality property, the time extent over which *T* operates, the possible decompositions of *T* into elementary step operators, the relationship between *T* and the Hamiltonian of the QTM, the properties that *T* should have in order to guarantee certain properties of the path that the QTM takes through Hilbert space, and so on. The art of QTMs (as with classical TMs) lies greatly in the art of picking the correct definitions for the model. We will not discuss QTMs further in this document, since although they are extremely interesting quantum mechanical systems, they are not very practical for considering how to design and implement computational algorithms.

II.2. Quantum Gate Arrays

In a classical computer, signals represented by voltages pass through logical gates and are relayed via metallization wires across a silicon chip. High voltages represent 1's, and low voltages represent 0's; storing a bit is accomplished by charging a capacitor with the appropriate polarity. In a quantum gate array, however, the qubits (various spins on a strand of a polymer, ions in an ion trap, photons in a pure state) stand still, and logical gates (selective radiofrequency pulses, coherent laser radiation, beamsplitters and phase-shift media) are applied, thus changing the states of individual qubits and modulating their interactions with one another. Readout of the result of a computation is accomplished at the end of the experiment, by performing a measurement (recording radiofrequency emissions, stimulating fluorescence via quantum-jump methods, interferometrically determining the phase of a photon) on the quantum system.

This is a very practical architecture for quantum computation: all that is required for implementing quantum computation on a particular physical system is to determine the most easily implemented quantum mechanical operations (the *elementary gates*) for that system, and then to compile the desired algorithm down to those gates. The computation can be executed in a finite amount of time and often is readily analyzable. Furthermore, this paradigm can be generalized to encompass the entire art of performing computation on natural materials, as exemplified by the tag project (discussed briefly in **VI.2.i.**).

The Toffoli gate (a three-bit reversible gate, which flips the third bit if the first two are both 1) is universal for quantum computation, in the sense that appropriate applications of the Toffoli gate, combined with single-qubit rotations, suffice to implement any quantum computation (Deu89). Over a period of several years, it was shown that two-qubit gates were universal (that is, any

unitary matrix can be decomposed into a product of unitary transformations that operate in 2D subspaces of C^n), that there existed a finite set of two-qubit gates that was universal, that there existed a single two-qubit gate that was universal (Bar95) (Bar95b), and finally that 'almost any' two-bit gate is universal (Llo94) (Deu95). The proof of the last claim can be accomplished by appealing to theorems about Lie algebras, or by explicitly showing via matrix dimensionality arguments that the set of non-universal gates has measure zero. To be explicit, any two-qubit gate which is specified by Euler angles which are not rational multiples of π is sufficient to approximate any gate with arbitrary accuracy, merely by iterating the gate and using single-qubit phase rotations to introduce new information into the quantum system as appropriate. Note that there is no direct analogy to this universality result in classical logic.

In the quantum gate array coprocessor paradigm, one constructs gates using a classical computer, and then applies them to the qubits using appropriate hardware. Although it has been shown that almost any gate is universal, for convenience we will take as our elementary operations the CNOT gate and the set of single-qubit rotations. The $\mathrm{CNOT}_{a,b}$ gate flips qubit *b* if qubit *a* is 1, and leaves *b* alone if *a* is 0. The subscripts indicate the qubits operated upon by a multi-bit operation: indices before the comma specify qubits to be conditioned upon (if any), and indices after the comma specify qubits to be operated upon. The conditional form of a gate *U* is denoted as $CU_{a,b}$, and can be formally defined as $(1 - A^+A) + (A^+A)U_b$, where A^+ and A are, respectively, the raising and lowering operators for spin *a*. When we refer to operators in matrix form, we will order the states in lexicographic order for simplicity; this is a natural ordering since we often consider a set of qubits as a 'register' holding a binary representation of a number. For example, going across or down an operator matrix or a density matrix in a 3-qubit space, we encounter the states in the sequence 000, 001, 010, 011, 100, 101, 110, 111 – just as we would count in binary. The Kronecker-tensor product command `krqn` (**Appendix A**) assists with the necessary lexicographic bookkeeping. In this lexicographic matrix notation, for example, the NOT gate can be expressed simply as

$$\mathrm{NOT} = \sigma_x = \begin{bmatrix} & 1 \\ 1 & \end{bmatrix}$$

and the $\mathrm{CNOT}_{a,b}$ gate takes the form

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{bmatrix}.$$

One useful gate derived from the CNOT gate is the SWAP_{ab} gate, which completely swaps two qubits, transposing their quantum states. This is especially useful for CAs or quantum computers that possess only nearest-neighbor interactions, such that we need to bring together information from remote places, let them interact, and then restore them to their original locations. SWAP_{ab} is simply equal to

$$\mathrm{CNOT}_{a,b}\,\mathrm{CNOT}_{b,a}\,\mathrm{CNOT}_{a,b}\,\text{(Bar95c)}.$$

A useful gate is the generalized Toffoli gate $\mathrm{C}^k\mathrm{NOT}_{a\ldots k,n}$ which flips qubit *n* if qubits *a*, *b*, . . . *k* are all 1.

Since we are using spins, names for commonly used SU_2 matrices can be very convenient, especially in the context of NMR experimentation. Code for these operators is provided in Matlab form, along with supporting and example code, in **Appendix B**. We define the SO_2 single-qubit rotation

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

Applied to a qubit in the state $(a, b) = a|0\rangle + b|1\rangle$, the resultant state is

$$(a\cos\theta + b\sin\theta)|0\rangle + (-a\sin\theta + b\cos\theta)|1\rangle.$$

We also define the single-qubit global phase shift,

$$P(\theta) = e^{i\theta} = \begin{bmatrix} e^{i\theta} & \\ & e^{i\theta} \end{bmatrix},$$

which is not directly observable, but which is a useful notational construct for making more complicated gates. In particular, a conditional global phase shift *is* observable, and is an essential component of many quantum computations.

In terms of the Dirac matrices, σ_x, σ_y, and σ_z (defined in section **I.1.i.**), we define specific rotations by θ about the *x*, *y*, and *z* axes, which correspond to expressions of the form $e^{i\theta\sigma/2}$. Let us denote these respective rotations by $X(\theta)$, $Y(\theta)$, and $Z(\theta)$. In the SU_2 (spinor) representation, using the definition of the matrix exponential immediately provides these operators with the explicit forms

$$X(\theta) = \begin{bmatrix} \cos(\theta/2) & i\sin(\theta/2) \\ i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix},$$

$$Y(\theta) = R(\theta/2) = \begin{bmatrix} \cos(\theta/2) & \sin(\theta/2) \\ -\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}, \text{ and}$$

$$Z(\theta) = \begin{bmatrix} e^{i\theta/2} & \\ & e^{-i\theta/2} \end{bmatrix}.$$

(Note that a rotation by 2π does not restore a spin-1/2 particle to its original condition; a 4π rotation is required to do this. In general, a spin-*n* particle is invariant under rotations by $2\pi/n$; this mysterious phenomenon is discussed in any good quantum mechanics textbook (Lib) (Sak).) Equivalent to the $Z(\theta)$ operator, up to a global unobservable phase $P(\theta/2)$, is the relative phase shift gate,

$$V(\theta) = \begin{bmatrix} 1 & \\ & e^{i\theta} \end{bmatrix} = Z(-\theta)P(\theta/2).$$

It is clear that $CP_{a,b}(\theta) = V_a(\theta)$. The relative phase shift gate is mostly useful for constructing the conditional relative phase shift,

$$CV_{a,b}(\theta) = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & e^{i\theta} \end{bmatrix}$$

which phase-shifts qubit *b* if qubit *a* is in the 1 state. The conditional relative phase shift is an very natural gate for implementing quantum computation on a linear ion trap (although it is less useful as a primitive in NMR quantum computation). For NMR quantum computation, the fundamental two-qubit computational gate is the J-coupling operator, which is the state-evolution operator corresponding to the scalar coupling of two spins for some period of time. The scalar coupling is due to the $I_z S_z$ term of the Hamiltonian (see **IV.1.ii.**): denoted *ZZ*, this operator takes the form

$$ZZ_{ab}(\theta) = \begin{bmatrix} e^{i\theta/2} & & & \\ & e^{-i\theta/2} & & \\ & & e^{-i\theta/2} & \\ & & & e^{i\theta/2} \end{bmatrix},$$

where θ is the effective angle that each spin rotates during the coupling period. If the J-coupling frequency is $2f_J$ Hertz, then $ZZ_{ab}(\theta)$ corresponds to the time-evolution operator that describes the passage of $\theta/(2\pi f_J)$ seconds. In particular, $\theta = \pi/2$, or the passage of $1/4f_J$ seconds, is sufficient to perform a CNOT operation. Some decompositions of $\mathrm{CNOT}_{a,b}$ into single-qubit rotations, J-coupling operators, and other primitives follow: $\mathrm{CNOT}_{a,b} =$

$$Y_b(-\pi/2)Z_a(-\pi/2)Z_b(-\pi/2)ZZ_{ab}(\pi/2)Y_b(\pi/2) =$$
$$Y_a(\pi/2)X_a(-\pi/2)Y_a(-\pi/2)X_b(-\pi/2)Y_b(-\pi/2)ZZ_{ab}(\pi/2)Y_b(\pi/2) =$$
$$Y_b(-\pi/2)CV_{a,b}(\pi)Y_b(-\pi/2)$$

By applying the rules of commutation for SU_2 matrices, many other equivalent pulse sequences can be designed. For some quantum computation experiments, a particular sequence may be preferred.

The Walsh-Hadamard matrix

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

is a very useful matrix: it allows us to change bases between the qubit and Bell bases, and it also is a common first step in a quantum computation, since it lets us prepare homogeneously-weighted states from a pure state. In general, the $2^n \times 2^n$ Hadamard matrix H^n has matrix entries

$$H^n_{x,y} = 2^{-n/2}(-1)^{\vec{x}\cdot\vec{y}},$$

where $\vec{x}$, $\vec{y}$ are vectors corresponding to the digits of *x*, *y* written in binary. For example, the entry in the 3rd row and 5th column of H^7 is

$$2^{-7/2}(-1)^{0000011\cdot 0000101} = -2^{7/2}.$$

Useful variants on the Walsh-Hadamard matrix include the matrix

$$U = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = Y(\pi/2)$$

which has the nice property that it can be implemented with a single NMR pulse. (We remark upon one notational caveat: the letter *U* often is used in the rest of this paper as a variable which represents an arbitrary gate; the meaning of *U* will always be apparent from the context.) From these primitives we will be able to build up a great many useful gates.

Many useful relations hold between these matrices, which may simplify the compilation of certain pulse sequences into forms natural for NMR quantum computation: for example,

$$X(\theta) = Z(\pi/2)Y(\theta)Z(-\pi/2)$$
$$H = Y(\pi/4)X(\pi)Y(-\pi/4) = Y(\pi/2)X(\pi) = UX(\pi)$$
$$Z(\theta) = Y(\pi/2)X(\theta)Y(-\pi/2) = X(-\pi/2)Y(\theta)X(\pi/2)$$

The last formula is especially useful, since Z pulses can't be directly applied by a transverse coil, and if one just waits for the spin to precess about the z-axis, the other spins change state during that time as well. It is much better to apply three quick deterministic pulses to effect the Z rotation. As suggested by these formulas, using conjugate representations is often a useful trick for designing pulse sequences: it can help to subtly modify the operation performed by a certain matrix, cause the action of an operator to be applied along a different axis, reduce the complexity of a pulse sequence at the expense of introducing some (irrelevant) global phase factors, or cancel other pulses in a sequence. Here are some examples of formulas inspired by conjugate representations, which might be instructive and useful:

$$H_a H_b \mathrm{CNOT}_{a,b} H_a H_b = \mathrm{CNOT}_{b,a}$$
$$\sigma_y = \mathrm{Z}(\pi/2)Y(2\pi)\mathrm{Z}(\pi/2)\sigma_x$$
$$\sigma_z = Y(\pi)\sigma_x$$

For a slightly more complicated example of how one might represent a complicated computation in terms of these elementary gates, consider the in-place Fourier transform on an L-qubit 'register' x, consisting of the physical qubits $i_1 \ldots i_L$ (so that the register provides the binary representation of an integer, or superpositions of such integers):

$$FT|x\rangle = \frac{1}{2^{L/2}} \sum_{y=0}^{2^L-1} e^{2\pi i xy/2^L} |y\rangle .$$

We can write this useful operation in terms of the gates U and $CV_{a,b}$ as follows (Pre96):

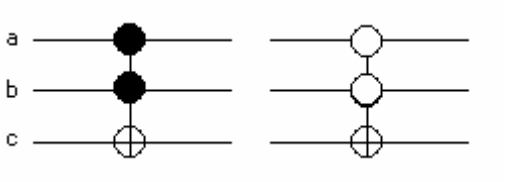
$$FT|i_1,...i_n\rangle = \prod_{j=0}^{L-1} (U_j \prod_{k=j+1}^{L} CV_{j,k}\left(\frac{\pi}{2^{k-j}}\right))$$

which takes only $L^2/2+L/2$ gates: L one-qubit gates and $L(L\text{-}1)/2$ two-qubit gates. This implementation unfortunately results in the qubits of the answer lying in reverse order within the register x, but this just requires mental relabeling of the qubits (see **V.1.i**. for a discussion of labeling and keeping track of information in quantum 'registers'). Using this Fourier transform, we now propose an operation which adds 1 (mod 2^L) to an L-qubit 'register' x of qubits. Perform FT on the register x. Then, for each qubit in the register (numbered in increasing order, k = 0, ..., L - 1, from right to left), apply the operator $Z(\exp(2^{k+1-N}\pi i))$ to the k^{th} qubit. After performing these L single-qubit operations, perform the inverse Fourier transform on the register x. The result is modulo-2^L addition on the register x, $ADD_x(1)$; a quick proof can be found in (Chu98a). We will make use of this function later.

We often want to make a unconditional particular operator into a conditional one, since this is a natural way to implement multiplexing, selecting, and enable functions, as commonly used in digital logic design. In general, any unitary operator U_b can be made into a conditional operator $CU_{a,b}$, e.g. an operator that only acts if qubit a is 1, by constructing the operator

$$CU_{a,b} = (1 - A^+A) + A^+AU_b$$

since A^+A, the 'number operator' (as defined in **I.i.i.**), returns 1 when a equals $|1\rangle$ and zero when a equals $|0\rangle$. The following notation for conditional gates, invented by Feynman, denotes on the left a conditional gate $C^2NOT_{ab,c}$ and on the right a conditional gate $C^2{}_0NOT_{ab,c}$ (the 0 subscript indicates that this gate flips c if a and b are both $|0\rangle$):



Three-bit gates are historically interesting, and are also important for understanding iterative methods for decomposing higher-order gates into CNOTs and single-qubit rotations. Empirically, (DiV94) found that numerically most 3-bit gates seemed to be approximatable with 6 two-qubit gates; in (Cha95) it was shown that the three-qubit Fredkin gate (a classical gate interesting to the reversible computation community), expressed in terms of single-qubit raising and lowering operators as

$$F_{abc} = I + a^+a(b^+c + c^+b - b^+b - c^+c + 2b^+bc^+c),$$

could be represented in terms of two-body operators as

$$CNOT_{c,b}NOT_cP_{ab,}(\pi/2)Q_{a,c}S_{b,c}Q_{a,c}S_{b,c}NOT_cCNOT_{c,b}$$

where $Q_{a,c}$ performs the operation σ_y on c when a is 1, $S_{b,c}$ performs the operation $(\sigma_y + \sigma_z)/\sqrt{2}$ on c when b is 1, and $P_{ab,}$ performs a conditional global phase shift, and therefore doesn't act on any specific qubit although it does condition on the qubits a and b. With a little extra work, one can reduce this expression to six two-qubit gates since several of the terms can be combined. The Toffoli gate $C^2NOT_{ab,c}$, also known as the doubly controlled NOT, can be implemented with 5 gates as

$$P_{ab}(3\pi/2)Q_{a,c}S_{b,c}Q_{a,c}S_{b,c}$$

where the P operation can be neglected (since the overall phase doesn't matter). Converting the Toffoli gate fully to CNOTs and single-qubit rotations takes a bit more work. If we let $S_{b,c}$ be the two-qubit gate that operates on c with $(\cos(\lambda/2)\sigma_y + \sin(\lambda/2)\sigma_z)$ when b is 1, then the above expression gives

$$\begin{bmatrix} I_6 & \\ & \cos\lambda + i\sin\lambda\sigma_x \end{bmatrix}$$

and in this spirit, many more complicated gates can made (Cha95). If we can ignore the relative phases of certain qubits, we can derive more efficient gate decompositions, although the resultant networks may not have strictly the same behavior as the specified computation. Nevertheless, such decompositions may be useful if a certain phase relation is not observable, or if we can follow the pseudogate by another suitably chosen one so that any net phase errors are cancelled. For example, we can construct a pseudo-Toffoli gate using only CNOTs and single-qubit rotations, as

$$Y_c(\pi/4)CNOT_{b,c}Y_c(\pi/4)CNOT_{a,c}Y_c(-\pi/4)CNOT_{b,c}Y_c(-\pi/4)$$

which results in the matrix

$$\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & -1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{bmatrix},$$

which differs from the real Toffoli gate only in the sign of one matrix element. This analysis may currently seem somewhat contrived and ad hoc, so we will reconsider three-bit networks after formalizing some of our observations.

At this point we will begin a discussion of higher-order gates, in an attempt to determine some systematic ways of analyzing them. The $C^k\text{NOT}_{a\ldots k,n}$ operation (which flips qubit n if the k qubits $a, \ldots, k$ are all equal to 1), for example, can be generalized to the C^kU operation,

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ldots & & \\ & & & u_{00} & u_{01} \\ & & & u_{10} & u_{11} \end{bmatrix}$$

which applies operator U to the last qubit if the k conditioned qubits are all 1. In the next few paragraphs we will show how to systematically reduce all such 'complicated', multi-qubit operations to operations on just a few qubits.

We will first make some general statements about matrices in the 2-dimensional unitary group U_2. First, note that any matrix in U_2 can be expressed as

$$P(d)Z(a)Y(\theta)Z(b),$$

with $d = 0$ if the matrix is in SU_2 (this discussion follows (Bar95), an excellent reference for understanding how to break down complex gates into simpler ones). This implies that every matrix U in SU_2 can be expressed as $A\sigma_xB\sigma_xC$ where $ABC = 1$ (proof: set $A = Z(a)Y(\theta/2)$, $B = Y(-\theta/2)Z(-(a + b)/2)$, $C = Z((b – a)/2)$, and evaluate these products directly). In particular, since σ_x equals the gate NOT, this implies that the conditional gate $CU_{a,b}$ can be written as

$$A_b\text{CNOT}_{a,b}B_b\text{CNOT}_{a,b}C_b$$

where $A\sigma_xB\sigma_xC = U$. (This generalizes; the $C^kU_{a..jk,n}$ gate can be written as

$$CA_{k,n}C^{k-1}\text{NOT}_{a..j,n}CB_{k,n}C^{k-1}\text{NOT}_{a..j,n}CC_{k,n},$$

and we will show how to implement $C^k\text{NOT}$ in $O(k)$ elementary operations. Therefore implementing C^kU also requires only $O(k)$ elementary operations.) Since any conditional SU_2 gate can be implemented using 5 elementary operations, and a conditional phase shift can be expressed as an unconditional relative phase shift (which in turn is equivalent to a Z rotation), it follows that any unitary conditional gate $CU_{a,b}$ can be expressed as the product of no more than 6 elementary operations.

Some special cases are included in the useful reference (Bar95). We summarize them here for their usefulness in interpreting and constructing NMR pulse sequences. In particular, we consider unitary operations which can be expressed as products of rotations through Euler angles, since those are very common. Occasionally it can be useful to use quaternion notation to evaluate the composite product of many rotations; this method is explained in (Blu85). For the SU_2 matrix

$$U = Z(a)Y(\theta)Z(a),$$

$CU_{a,b}$ can be written as

$$A_b\text{CNOT}_{a,b}B_b\text{CNOT}_{a,b}$$

where

$$A = Z(a)Y(\theta/2),\ B = Y(-\theta/2)Z(-a),$$

and for

$$U = Z(a)Y(\theta)Z(a)\sigma_x,$$

$CU_{a,b}$ can be written simply as

$$A_b\text{CNOT}_{a,b}B_b.$$

For C^kNOT gates, simplification is possible if extra scratch qubits (which are completely restored to their original states after the computation finishes) are available: for $k \leq \lceil n/2 \rceil$, a C^kNOT gate can be simulated using $4(k - 2)$ C^2NOT gates. If our qubits are labeled 1, 2, …, k, …, n, then the decomposition is

$$\Delta^* \text{C}^2\text{NOT}_{1\ 2,\ n-k+2}\ \Delta\ \text{C}^2\text{NOT}_{k\ n-1,n}\ \Delta^*\ \text{C}^2\text{NOT}_{1\ 2,\ n-k+2}\ \Delta\ \text{C}^2\text{NOT}_{k\ n-1,n}$$

where Δ is,

$$\text{C}^2\text{NOT}_{3\ n-k+2,\ n-k+3}\ldots\text{C}^2\text{NOT}_{k-2\ n-3,\ n-2}\text{C}^2\text{NOT}_{k-1\ n-2,n-1}.$$

The first part of the sequence negates the i^{th} bit if the first $i - (n - m)$ bits are 1, and the second part reverses all the transformations except for the possible negation of bit n. Note that no special preparation or postprocessing is necessary for the ancilla spins – the bits $k + 1$, …, $n - 1$ can have any value, and are returned to that value at the end of the computation.

Some other useful results, which are useful in specific cases (Bar95), are listed below for reference:

- For $m \leq k - 1$, with qubits labeled 1, 2, …, k, …, $k + 2$ (that is, using one extra scratch bit),
  $$\text{C}^k\text{NOT}_{1\ldots k,k+2} = \text{C}^{k-m+1}\text{NOT}_{m+1..k+1,k+2}\text{C}^m\text{NOT}_{1..m,k+1}\text{C}^{k-m+1}\text{NOT}_{m+1..k+1,k+2}\text{C}^m\text{NOT}_{1..m,k+1}$$
  Due to cancellations and repeated operations, many of these suboperations need only be conducted modulo phase shifts, resulting in the fact that in this construction takes at most $48k - 108$ elementary gates. As noted previously, a $C^kU_{a..jk,n}$ gate can be written as
  $$CA_{k,n}\text{C}^{k-1}\text{NOT}_{a..j,n}CB_{k,n}\text{C}^{k-1}\text{NOT}_{a..j,n}CC_{k,n},$$
  where A, B, and C are appropriately calculated from U.
- If one has no spare bits, then the best one can do is express C^kU in $O(48k^2)$ operations, via the decomposition
  $$C^kU_{1..k,k+1} = C^{k-1}V_{1..k-1,k+1}\text{C}^{k-1}\text{NOT}_{1..k-1,k}CV^*_{k,k+1}\text{C}^{k-1}\text{NOT}_{1..k-1,k}CV_{k,k+1}$$
  where $V^2 = U$ – the proof is clear by the methods suggested above.

The number of 2-qubit gates required to simulate a general n-qubit gate can be found by dimension counting, suggesting a lower bound of $O(4^n)$. This means that most unitary quantum

computations are intractable for implementation on an NMR quantum computer. (Of course, most quantum computations don't do anything.) Interestingly, one can *approximate* a unitary operator C^kU by O($k \log(1/\varepsilon)$) operations, where ε is the Euclidean norm of the vector difference between the real result and the actual result of a unitary computation (Cop94). The proof relies on the fact that in the two recursive methods described above, where in recursion stage $i+1$, $V_{i+1}^2 = V_i$, the phases in step i are proportional to $1/2^i$, and therefore can be neglected after O($\log(1/\varepsilon)$) steps.

II.3. Interpreting the Quantum computer

II.3.i. Proposals for quantum computers

There have been many proposals for systems capable of quantum computation. Svozil proposed a system which uses the phases of photons to encode qubits (Svo94). He demonstrates how the standard "toolbox" of laboratory optical devices gives a complete set of quantum computing primitives – i.e., beam splitters and partially-transmitting media can effect single-bit unitary transformations, phase shifters can change the phase of quantum states, and parametric up- and down- conversion can be used to create or consolidate multiple quanta. Unfortunately the simultaneity required to do multiple-photon manipulation is fairly sophisticated, and thus this proposal is rather ineffective for computation (although for transmission of quantum systems, as in quantum teleportation, optical techniques are becoming increasingly mature).

One proposal of Cirac and Zoller is the use of a linear harmonic ion trap to perform quantum computation (Cir95). It consists of a collection of localized ions in an ultrahigh vacuum, each with a ground state and multiple excited states. Due to their isolation, each excited ion state is relatively stable. The advantage of such a device is that there is a 'global register' in contact with all the ions – namely, the normal modes of the trap due to Coulomb repulsion between the ions – which allows the entire lattice to be excited when a single ion absorbs or emits a photon, much like a discrete version of the Mossbauer effect in a crystal. This means, for example, that localization of ions is not a limiting factor in the types of operations which can be performed (Pre96). For example, a C^kNOT operation (discussed in the above section) can be performed in a mere $2k+3$ laser pulses, by using the lowest trap phonon mode as a transference register. By comparison, NMR has no such global register, and a C^kNOT is very expensive in terms of the natural primitives of NMR computation. Unfortunately linear ion traps are very difficult to construct and maintain, and to this day no significant progress has been made in constructing a feasible quantum computer with one.

Another proposal, which may have better scaling properties than even NMR, is that of using quantum dots – (DiV97b) has proposed that electronic spins in quantum dots (which are commonly called 'artificial atoms') could be used as qubits, although this technology is still in its infancy, and nobody has shown demonstrated how to control interactions between different quantum states in this paradigm. Common buzzwords amongst people in this camp include 'spin valve,' 'single electron transistor,' 'ballistic transport,' and 'superconducting' – certainly there are many viable technologies out there, but none with experimental capabilities along the lines of quantum computation.

One very specific proposal that is appealing is the use of isolated phosphorus spins in isotopically pure silicon (Kan). A single such spin has an excited-state lifetime that could approach the age of the universe, since there is no lattice of spins with which to couple and relax (the most common isotope of silicon, like that of carbon, possesses a spin-zero nucleus). Certain gates could be turned on and off, allowing nearby spins to interact by lowering an effective energy barrier. Like the above proposals, this is highly speculative, although potentially highly scalable.

But as for liquid-phase NMR – it works. Nothing else has come close. NMR is not a perfect substrate for quantum computation, but it gets the job done. And it has lots of applications to chemistry, biology, material science, and other disciplines. So that's why we're in the NMR business.

II.3.ii. Beyond the coprocessor paradigm

One interesting architecture for a quantum computer has been suggested by Benioff, that of quantum robots (Ben97). A quantum robot might be formed by separating a quantum Turing machine's processor module from its tape, and calling the former the 'robot' and the latter its 'environment'. The 'robot' would perhaps have its own ancillary memory and output system (i.e. an onboard Turing machine), and it could both affect its environment (say, by flipping spins on a lattice) and be affected by it (e.g., by collecting quantum data). A simple action-response mechanism is outlined by Benioff in his paper, although it is uncertain whether such a quantum robot could do anything useful or meaningful to human beings. One suggestion that he makes is that, since all physics experiments are essentially quantum-mechanical, such a robot could explore the language (in the computability-theory sense) of all physically meaningful experiments. It is my opinion that scientists will have plenty of time to think about the uses of such machines before they actually become available.

More modestly and perhaps even realistically, it has been proposed that a quantum computer could execute a quantum program with a fairly classical-looking architecture; however this possibility is fraught with difficulty in that it is not clear where or when human observation could occur. Following the discussion in (Nie97), assume that a quantum computer capable of universal computation starts in the state

$$|d\rangle \otimes |P\rangle,$$

with program P running on data d. Suppose that on each clock tick the fixed operator G (which one might think of as a universal Turing machine operator) is applied to the computer, mapping, say,

$$G : |d\rangle \otimes |P_U\rangle \rightarrow U|d\rangle \otimes |P_U'\rangle$$

where the next state of the program P_U' does not depend on d. Although such a mapping is fairly easy to implement for a classical computer (as anyone who has written microcode will acknowledge), it can be shown that for the quantum case, it is impossible to execute every possible program U, given any particular operator G – in other words, the quantum equivalent of the universal Turing machine can't be implemented in a model this simple. In fact, for an N-dimensional Hilbert space, only N distinct programs U can be implemented. The proof is in the

same spirit as the no-cloning theorem: suppose that operators U_p, U_q are encoded in program registers $\left|P_p\right\rangle$ and $\left|P_q\right\rangle$, respectively, and apply G to these two states of the computer, starting with the same arbitrary data d in each case. Taking the inner product of these results gives

$$\left\langle P_q \middle| P_p \right\rangle = \left\langle P_q' \middle| P_p' \right\rangle \left\langle d \middle| U_q{}^* U_p \middle| d \right\rangle,$$

which implies for nonzero $\left\langle P_q' \middle| P_p' \right\rangle$, that $\left\langle d \middle| U_q{}^* U_p \middle| d \right\rangle$ has no d-dependence. Therefore $U_q{}^* U_p$ must be the identity, and so U_p and U_q must be identical (up to a global phase). This is a contradiction, so $\left\langle P_q' \middle| P_p' \right\rangle$ must be zero, and thus the programs must be orthogonal.

The authors of (Nie97) suggest a probabilistic way to implement such a universal gate array, but the probability of an arbitrary computation succeeding is 2^{-2m}, where m is the number of data bits, and so there is no gain associated with using such a quantum computer.

Finally, a curious note on diagonalization arguments and quantum computation: as it turns out, the diagonalization procedure used to prove the undecidability of the halting problem does not *directly* apply to quantum computers, which operate in a superposition of bit states. A diagonalization argument like that given in part **I.1.iii.b**. – which effectively swaps 0 and 1 – has a solution in Hilbert space, namely the fixed-point of whichever unitary operator is being used in the diagonalization argument (Svo94). For example, [0 1; 1 0], the 1-qubit NOT operator, has a fixed point, the symmetric Bell state. Of course, any irreversible measurement collapses the fixed point state, thus giving back classical undecidability, since the measurement results in a definite value and leaves the quantum system in an eigenstate of the measurement operator, which is vulnerable to diagonalization arguments. But as long as you don't look, the undecidability of the halting problem is not provable in quantum recursion theory, as far as we know. However, you might have to have a quantum-mechanical consciousness to understand the answer. We will halt this discussion now before we enter a superposition state of fruitlessness and idiocy.

II.4 Quantum Cellular Automata

Creating large systems of quantum particles with controllable interactions which are isolated from their environment is a difficult challenge, especially when one considers the equipment needed to address the various quantum components. One suggestion for avoiding this situation is to create *quantum cellular automata* – lattices of identical spins (or small groups of spins) which interact with their neighbors, allowing computation to take place. In **III.1.i.** we examine a paradigm in which this is directly applicable to a quantum computation, namely that of simulating a lattice gas of particles with quantum nature.

A *cellular automaton,* as defined by John von Neumann and explored to great detail by Wolfram (Wol94), Boghosian (Bog97), and a great many other people in the last few decades, is a lattice of sites together with an operator ϕ which maps the current lattice state L and the current time to a new state L'. One common subset of cellular automata is that of *local additive cellular automata,* in which

$$\phi_{t+1}(x) = \sum_{e \in E(t,x)} w(t, x+e)\phi_t(x+e)$$

where E is some set of 'neighbor' lattice vectors (Mey96) – usually taken to be d orthogonal vectors in R^d. If E, w are time-invariant and E is also translation-invariant, we say that the cellular automata is *homogeneous*. If the weights w make up a doubly stochastic matrix, then $\phi_t(x)$ can be interpreted as the probability of a system being in state x at time t. If w is unitary and the state value is complex, then the CA is called *scalar unitary*. An amazing fact is that homogeneous scalar unitary CAs can only evolve by constant translation along the lattice, or by a change of phase. The proof, in brief, is based on the observation that the unitary evolution operator for a lattice with nearby-neighbor interactions is band-diagonal, and the only band-diagonal solution to the unitarity requirement $UU^* = 1$ is a translation matrix with an arbitrary overall phase. Simply put, homogeneity is a very boring restriction; restricting the evolution matrix to be invariant only under some subgroup of the lattice vectors is a much more modest restriction. Additional essays on quantum cellular automata are given in (Mey96) (Mey96c).

A practical use of quantum cellular automata for quantum computation was suggested by (Llo93). In reality, it is much easier to create long polymers with identical repeated subunits than to create a long molecule with completely distinct spins along the length of the molecule. Especially for NMR quantum computation, where the atoms neighboring a certain spin determine the ease of performing computation on that spin, addressing many distinct spins with slightly different chemical shifts may be difficult or impossible; having a long chain of identical spins would make addressing a great many qubits easier, albeit at a cost due to the possibly more complicated spin manipulations. In light of this observation, consider a chain of species A'BCABCABC… where the Larmor frequencies for A, B, and C are different, and where nearest-neighbor interactions are given by the J-couplings J_{AB}, J_{AC}, and J_{BC}. The first A is different because there is no C to its left, which is important because otherwise there would be no unambiguous way to read out information from the quantum spin system. One can program the ensemble by, say, applying a soft pulse $\pi(\omega_{01}{}^B)$, which rotates spin B by 180^o only if spin A is in state 0 and C is in state 1. Compare this with a classical CA rule (see the above definition or Wol94): in each case, many identical spins change state depending on the states of their neighbors. To execute $CNOT_{C,B}$, which flips each B if the adjacent C is in the 1 state, simply execute $\pi(\omega_{01}{}^B)\ \pi(\omega_{11}{}^B)$ – two soft pulses are required so that B is flipped regardless of the state of A. To swap the state of all the B's and all the C's, we can use the decomposition noted in **II.2.** to write $SWAP_{B.C}$ as

$$\pi(\omega_{01}{}^B)\ \pi(\omega_{11}{}^B)\ \pi(\omega_{10}{}^C)\ \pi(\omega_{11}{}^C)\ \pi(\omega_{01}{}^B)\ \pi(\omega_{11}{}^B).$$

By addressing the A' spin, we can load a quantum bit into the polymer, assuming that it is in a known state. We can then swap A' and B, then B and C, then C and A, and so on, until the loaded data is where we want it to be. We then can load the next bit into A', and perform more operations to locate it in its proper place. Of course, now we have lost the original bit that we loaded into A' in the first place – properly using the quantum CA/SIMD processor takes some care.

We briefly note that it is possible to perform a Grover's search on n qubits (2^n strings in the database) without losing the square-root speedup offered by Grover's algorithm, if the n qubits are in the CA form A'BCABCABC… The process is as follows: use four Hadamard transforms

(A', A, B, C) to tilt each spin into the Bell basis, and then iterate $2^{n/2}$ times the operation *CD* (as described in **III.1.iii.**), appropriately decomposed into single-qubit rotations and CNOTs (as described in **II.2.**). Whenever a single-qubit rotation on a spin is required, perform SWAPs until that spin state is located on the nucleus A', then execute the rotation on A', and finally reverse the order of the SWAPs until that state has been restored to its original location. Whenever a CNOT between two spins is required, move one spin state onto A' and the other into the first B, execute a CNOT, and then reverse all the SWAPs which were used for relocation. It is trivial to see that this simple procedure, which requires no bookkeeping for locating any of the information in the CA, can carry out Grover's algorithm with at most a logarithmic increase in the number of execution steps, since the overhead for using a CA model is merely a polynomial function in the number of qubits *n*.

II.5. Quantum chaos and dynamics

Proposals have been made to study 'chaotic structures' using quantum mechanics. One can quantize the Baker's map (mentioned in **I.i.vi.**), for example, and study the dynamics of the resultant system as follows (Bru98). Define position and momentum operators *p*, *q* such that $[p, q] = 2\pi i/N$, where $N = 2^n$ is the dimension of the Hilbert space, and so that each of *p* and *q* has eigenvalues $1/N, \ldots, (N\text{-}1)/N$. The discrete Fourier transform *FT* (see **II.2.**) transforms from the momentum basis to the position basis, and vice versa. As we have done many times in the past by this point, we can associate with a basis state

$$\left|q_s\right\rangle = \bigotimes_{i=1}^{n} \left|a_i\right\rangle$$

a binary number $q_s = 0.a_n \ldots a_1$. The quantum Baker's map can then be defined as

$$B = FT_{0..n}{}^{-1}(I_n \otimes FT_{0..n-1}),$$

which equates to first performing the Fourier transform on the least significant n - 1 qubits (leaving the most significant qubit alone), then performing the inverse Fourier transform on the entire set of qubits.

By comparing

$$FT_{0..n}{}^{-1} \bigotimes_{i=1}^{n} \left|a_i\right\rangle \text{ and } (I_n \otimes FT_{0..n-1})^{-1} \bigotimes_{i=1}^{n} \left|a_i\right\rangle,$$

one can see that $B\left|\psi\right\rangle$ does have some properties of a shift operator, which we will not discuss here. An NMR pulse sequence for implementing the quantum Baker's map on trichloroethylene is given in (Bru98). The pulse sequence is as yet too long to perform on a real NMR spectrometer.

It has been hypothesized that a constant Kolmogorov-Sinai entropy (that is, the additional information about the initial condition of a dynamical system required to forecast the behavior of the system for an additional unit of time) amidst decoherence is a sign of quantum chaos (Zur94) (Ali96). The classical Baker's map, which shifts each bit down one level of significance after each iteration, has a Kolmogorov-Sinai entropy of 1 bit. For NMR, since different bits are encoded on different qubits, this means that different spins must decohere at different rates to simulate the increased effects of noise on less significant bits (e.g., the spins must have radically different T_2's). (If this were not the case, then less significant bits would not really be 'less

significant', since they could decohere less rapidly than the more significant bits!) Under the action of the quantum Baker's map, this process has been numerically simulated, showing that the von Neumann entropy of the system quickly converges to its maximum (Bru98). Unfortunately due to the fact that information is transferred back and forth between species with different T_2's in the process of the computation, and the fact that the T_2's are actually quite similar, the trichloroethylene simulation proves to be rather nonchaotic. (However, hypersensitivity to perturbations, another characteristic of chaos, was suggested by the simulations in (Bru98).)

"What is power? It is the power to do, to change, to be, to understand,
to freeze in fire, to hide behind glass, to scream in silence…"
– *E*

III. Power of Quantum Computation

III.1. Problems at which Quantum Computing is Good

III.1.i. Simulating Quantum Systems

Feynman's observation that classical computers require exponential investments of both space and time to simulate quantum systems was perhaps the first indication of the potential of quantum computation (Feyn82). Indeed, except for Shor's factoring algorithm, no other interesting algorithm is known to benefit from a speedup of exponential magnitude. It has been shown how a set of qubits might be used to simulate the time-evolution of an arbitrary many-body quantum system (Abr97). For example, to simulate a Fermion system of n particles, each of which can take m states, it suffices to use $n \log_2(m)$ qubits of a quantum computer, as opposed to $O(2^n)$ classical bits. The system, which is assumed to start in the second-quantized state

$$|00...0\rangle,$$

can be prepared in almost any interesting natural state (atoms on a lattice, thermal states, weakly interacting particles, etc.) in polynomial time. By applying an appropriately coded local unitary operator corresponding to the desired time-evolution operator, the system's evolution under a Hamiltonian can be simulated, although certain aspects of the Hamiltonian may require clever programming of 'scratch' qubits to maintain ancillary information. Readout takes place with respect to the orthogonal basis of some suitable observable; clearly it is impossible to see the actual wavefunction, so all information must be accumulated in terms of observed statistics. However, performing k experiments reduces the error in estimating a probability by a factor of $1/\sqrt{k}$, so the accuracy of an experiment is polynomial in the number of trials: thus, the entire procedure can be accomplished in polynomial time.

An antisymmetric (first-quantized) Fermi system simulator is also described in (Abr97); in this paper, the authors also present a detailed analysis of the Hubbard model of electrons on a lattice, and show how a simulation might be implemented on a first-quantized system in $O(n^2(\ln m)^2)$ operations.

One model which is interesting to physics-of-computation specialists is the lattice gas, a discrete system which replicates some fairly complicated fluid behavior. Many attempts have been made to derive all of physics from such discrete systems, due to the aesthetic qualities of such simple models. A lattice gas comprises 3 components: a regular lattice of N sites, a finite number of particles with well-defined interactions, and a clock. Every execution of the clock causes the particles to propagate so as to conserve momentum (or some other quantity); if the particles collide on a vertex of a square grid of a momentum-conserving lattice gas, for example, they will exit the vertex so as to conserve momentum. The d-dimensional quantum lattice gas (QLG) is similar, except that at each of the N sites there exist $2d$ qubits, each of which describes a quantum-mechanical amplitude moving forwards or backwards along one of the d lattice vectors (Bog97). The clock update then causes the complex amplitudes to stream according to their

associated velocity vectors, and then a $2^{2d} \times 2^{2d}$ *unitary* operator is applied to the $2d$ qubits at each lattice site. This causes entanglement and makes the problem exponentially hard to simulate on a classical computer.

For example, a 1-D lattice gas might be such that qubits can move left or right, with two qubits at each lattice site, and with a symmetry-preserving collision operator

$$T = \begin{bmatrix} 1 & & & \\ & q & p & \\ & p & q & \\ & & & \phi \end{bmatrix}$$

applied to each site. Note that in order to insure the unitarity of *T*, the entries must satisfy $|\phi| = 1$, $pq^* + p^* q = 0$, $|p|^2 + |q|^2 = 1$. Considering the one-dimensional subspace containing only one particle moving right or left, we can write down equations of motion for $\psi_r(x)$, $\psi_l(x)$ so that

$$\psi_r(x,t+1) = q\psi_r(x-1,t) + p\psi_l(x+1,t)$$
$$\psi_l(x,t+1) = q\psi_l(x+1,t) + p\psi_r(x-1,t)$$
$$\psi(x,t) = (p+q)^{-t}\left(\psi_l(x,t) + \psi_r(x,t)\right)$$

It can then be shown that in the continuum limit – e.g., taking the space increment *x* to go like ε and the time increment *t* to go like ε^2 – that $\psi(x,t)$ satisfies the free-particle Schrodinger equation for mass $m = ip/q$. (The mass is real, due to the unitarity constraints.)

Due to the discrete nature of the QLG, in order to model particles in an external potential one simply multiplies each qubit by

$$U = \begin{bmatrix} 1 & \\ & e^{-i\varepsilon^2 V(x)} \end{bmatrix}.$$

Simulations show that with even very few (~10) lattice sites, one can easily replicate the eigenstates of simple potentials like that of the simple harmonic oscillator. The speed with which a classical computer can simulate a *d*-dimensional QLG system with *L* lattice sites in each direction and *n* particles is of order $L^{Dn+2}(2d)^n/n!$, while a quantum computer could accomplish this task in order $(2D)L^{2+D}$ steps (Bog97).

A detailed study of QLGs is given in (Mey96b, Mey96d, Mey97). We will not go into this topic further here.

III.1.ii. Factoring

We very briefly outline Shor's factoring algorithm (Sho96). The idea behind factoring *n* is to find, given any integer *x*, the least integer *r* such that

$$x^r \equiv 1 (\mathrm{mod}\, n).$$

This is equivalent to solving for the period of the modular exponentiation function. If such a number could be found, then

$$(x^{r/2} - 1)(x^{r/2} + 1) = x^r - 1 \equiv 0 (\mathrm{mod}\, n)$$

and a factor of n has therefore also been found – unless r is odd or $x^{r/2} \equiv -1 (\mathrm{mod}\, n)$. It is elegantly proven in (Sho96) that these latter aberrant cases only occur with probability 2^{1-k}, where k is the number of distinct, odd, prime factors of n.

The algorithm works by setting up a first quantum register in the uniform superposition of all integers r (represented of course as binary strings), then performing modular exponentiation of a classical number x with an exponent equal to the quantum number stored in the first register. The answer is stored in a second register, which is therefore in a superposition of x^r (mod n) for all r. Performing a Fourier transform (which can be done in time polynomial in the number of qubits, on a quantum computer) then determines the period of the modular exponentiation function. The readout step, which is somewhat subtle and contains some very pretty number theory, is explicated beautifully in Shor's paper. No attempt to summarize the method is given here, given the difficult of improving upon the conciseness of his argument.

III.1.iii. Searching

Grover (Gro96) has discovered a search strategy which finds one of S items from N unsorted possibilities in $O(\sqrt{N/S})$ steps, while a classical search strategy would be expected to take $O(N/S)$ steps. This has been proven optimal – that is, quantum computation offers precisely a square-root speedup, and no more (Ben96b), for general unstructured search. Many people have tried to accomplish better results for structured searches (Hog97) (Hog98), but just as with their classical counterparts, the results are often ambiguous and the success of an algorithm heavily depends upon the specifics of a particular instance of the problem.

We treat our n-qubit system, which allows us to search for one of $N = 2^n$ items, as being in a state described by a binary string (or superposition of binary strings, as the case may be). We require a unitary search operator C which operates on the n qubits and rotates the phase of a string by 180^o if the string is the desired binary string S^*, and leaves the string alone otherwise. The algorithm is as follows: starting with the ground state, tip each qubit into the state

$$(|0\rangle + |1\rangle)/\sqrt{2}$$

using n applications of the single-qubit Walsh-Hadamard transform H (see **II.2**.), and then repeat the following sequence $2^{n/2}$ times: first apply the operator C to the system, then apply the 'inversion about the mean' operator D, also known as the diffusion matrix:

$$D = \begin{bmatrix} -1+2/N & 2/N & .... & 2/N \\ 2/N & -1+2/N & .... & ... \\ ... & ... & ... & 2/N \\ 2/N & ... & 2/N & -1+2/N \end{bmatrix} = H_n R H_n$$

where R is the matrix diag([1, -1, -1, …, -1]). (You can prove that D indeed equals $H_n R H_n$, by splitting the matrix R into $-I$ + diag([2, 0, 0, …, 0]) and writing out the explicit equations for each matrix element of D, via direct calculation using the definition of H_n.) Finally, read out the answer by measuring the state with respect to the qubit basis.

Grover's search algorithm works in the following way: the iterated loop redistributes the

amplitudes of each basis vector so that the amplitude of the desired string increases by $\sim N^{-1/2}$ after each cycle through the loop, and the amplitudes of all the other strings decrease by the appropriate amount. The matrix D acts so as to reflect the amplitude of each state over the mean amplitude m of all the states, so that a state with amplitude $m - a$ ends up with amplitude $m + a$. To see this, define P as $(D + I)/2$, so that $P_{ij} = 1/N$, and note that Pv replaces each component amplitude of the state v with the mean of all the components' amplitudes. It immediately follows that $Dv = -v+2Pv = Pv + (Pv - v)$, which precisely fits the intuitive picture of the process of inverting about the mean. Since C sends S^* to $-S^*$, while all the other binary vectors' amplitudes are preserved by C, during each iteration of the loop the amplitude of S^* therefore goes up by about twice the average ($\sim N^{-1/2}$) and all the other amplitudes decrease slightly. A more formal proof that the amplitude increases by at least $1/2\sqrt{N}$ in each application of D can be proved by simple arithmetic.

To see that one can not do better than $\Omega\left(\sqrt{N}\right)$ steps, one can prove a theorem which shows that any iterated quantum oracle algorithm that runs for t steps can depend on at most t^2 qubits (Ben97b) (Boy96) (Zal98). The argument is as follows: if the algorithm runs for less than t steps, then there exists an oracle C that returns results that cannot be statistically told apart from those of the null oracle C_0 (i.e., the identity matrix which doesn't change the phase of any string). One can see this fact by writing out the difference $\phi - \phi_0$ between the states computed by the two oracles, in terms of the difference $C - C_0$ of the two oracle matrices, then summing the amplitude of this difference over all N strings that could have been selected by the oracle C. The final step of the proof is showing that this sum, which provides a measure of the distinguishability of the two resultant distributions, is at most proportional to the number of time steps squared. This implies that in less than $O(t^2)$ steps, not enough amplitude concentration occurs for two oracles to be told apart with any statistical significance, and the search must of necessity fail.

The algorithm has been extended to sorted databases of N entries, or analogously a database which can answer many questions in parallel: in the classical case, $\log_2(N)$ stages are required (e.g., by searching using the binary-chop method), whereas *only a single query* (e.g., execution of an oracle operator) is needed for the quantum case (Gro97). There is a catch, however: $O(N \log(N))$ identical subsystems, as well as a proportional number of preparation and postprocessing steps, are required to prepare for the (single) rather complicated query. One can also extend Grover's search algorithm to arbitrary initial amplitudes – or at least one can exactly calculate the errors incurred by performing Grover's search on states with arbitrary initial amplitudes (Bir98).

An operator C which, when applied to an initial state results in a finite probability of arriving at a certain target state, can be assisted by a Grover-like amplitude enhancement algorithm. In general, Grover's search architecture can be adapted to make any such iterative computation converge to the target state, with the probability of a successful measurement growing *quadratically* in the number of iterations. This shows how to attain the famous square-root speedup in other situations which might be amenable to an amplitude-enhancement algorithm. This framework has been instantiated, for example, for the purpose of finding the mean (m) and the median (M) of numerical data distributions encoded in the amplitudes of a set of qubit strings (Gro97c). Classically, to find the m or M of a dataset to a precision ε requires $\Omega(\varepsilon^{-2})$ sampling

operations, whereas the analogous quantum mechanical method takes only $\Omega(\varepsilon^{-1})$ operations, a square root improvement in the amount of time required to achieve a certain error. The main point of the general amplitude enhancement architecture is that one can replace the Walsh-Hadamard transform by *any* unitary transformation U, as follows: define the operator V_a as $I - 2e_a e_a^t$ where e_a is the vector with coefficient 1 in the slot represented by the binary string a, so that $V_{a\,ij} = -1$ if $i = j = a$, 1 if $i = j \neq a$, and 0 otherwise. Then if a quantum system starts in state s and it is desired to perform a computation that ends with the system in state t, applying the operator

$$-V_s U^{-1} V_t U$$

preserves the vector space spanned by e_s and $U^{-1}e_t$, and after η iterations of this procedure, the state e_s evolves to

$$\cos(2\eta|U_{ts}|)\, e_s + \sin(2\eta|U_{ts}|)\, U^{-1}e_t,$$

where $U_{ts} = \langle t|U|s\rangle$. This amplitude is closest to the state $U^{-1}e_t$ when $\eta = \pi/4|U_{ts}|$; a single additional application of $-V_s U^{-1} V_t U$ then results in precisely the state e_t. Replacing U by H_n gives the standard search algorithm described at the beginning of this section, but there are many other possibilities. As an example of what can be done with this architecture, suppose you want to create an associative memory where words that differ by k bits are mapped to the same word. Then simply set U equal to

$$\bigotimes_{i=1}^{n} \begin{bmatrix} \sqrt{1-k/n} & \sqrt{k/n} \\ \sqrt{k/n} & -\sqrt{1-k/n} \end{bmatrix}_i$$

and iterate this operation $\sim 1/|U_{ts}| \sim (n/k\text{-}1)^{k/2}(1\text{-}k/n)^{n/2}$ times to get the associative match. Designing the U operation is often tricky, though; one might need to use calculus to optimize the entries of the U matrix with respect to the structure of the problem at hand, so that U_{ts} is as large as possible, and so that U_{ts} connects all the desired initial and final states with nonzero probability.

Attempts have been made to develop structured search algorithms on quantum systems (Gro98) (Far97) (Hog97). Brute-force search is almost never the right way to solve a problem – usually there is some regularizer, criterion, smoothness, or other knowledge about a problem which suggests some heuristics which can make the problem much easier. NP-complete problems often offer the possibility of building up complete solutions using partial solutions, depending on the parameters of the problem. Structured algorithms, being problem-dependent, exhibit a 'phase transition' between small and large problems, as noted earlier in this document (Kir94). The basic process for structured quantum search is to take the Grover algorithm, but to add one more step to each iteration, which performs a structure-based operation, changing the phases as necessary (e.g., leaving the amplitude of a string alone if it appears to be a nogood or an obviously nonoptimal good string, and inverting the phase if the string looks good) so as to move amplitude to the best good strings. Empirical simulation has shown that these methods exhibit the classical 'phase transition' and often fare better than unstructured quantum search, although as in the classical case, these claims remain theoretically unjustified (Hog97).

A specific proposition for accomplishing structured quantum search is *nested quantum search* (Cer98). This reduces the time complexity of the algorithm to $N^{a/2}$ where a is some constant depending on the tree structure of the problem, and N is the number of strings to be searched

over. A simple classical structured search algorithm, in which one tests the extensions of a given set of partial solutions (say, of a certain length), terminating obvious nogoods in the process, can be generalized to a quantum algorithm. Of course, one must choose when to stop the breadth-first generation of partial solutions (at some level i), and begin the depth-first search which eliminates wrong answers (from level i of the tree downwards to level $\log(N)$). An explicit strategy which accomplishes this is to use unstructured search to construct a state in a superposition of all possible solutions down to some level i of the solution tree, then performing unstructured quantum searches in each of the resultant $(\log(N) - i)$-dimensional solution subspaces simultaneously.

III.1.iv. Other Problems

Deutsch's problem asks whether a function f is 'constant' or 'balanced'. For example, there are four possible f's which take in one bit as input and return one bit as input: f can return 0 or 1 upon all possible inputs (a constant function), or it can return different values on different inputs (identity, NOT) (balanced). Determining the nature of f is equivalent to finding XOR($f(0)$, $f(1)$). Finding this exclusive or returns only one bit, but f must be evaluated twice – a most unsatisfying state of affairs. However, if one starts in the state

$$|0\rangle + |1\rangle \otimes |0\rangle - |1\rangle,$$

then applying a unitary version of f only once immediately gives the result

$$(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle \otimes |0\rangle - |1\rangle,$$

and checking the relative phase of the first register (e.g., by finding the polarization) immediately gives the answer. This problem has been realized experimentally, for the two-qubit case (Jon98), and was the first algorithm ever to be executed on an NMR quantum computer, using cytosine in heavy water (D_2O).

One speculative paper (Abr98) explains how if there were nonlinearities in quantum mechanics, NP complete problems and #P problems (e.g., problems with an oracle, in which the number of solutions must be established) would be solvable in polynomial time. One can think of this as a nonlinear form of Grover's search; the crucial element is a nonlinear operation with positive Lyapunov exponent over some finite region (say, a solid angle α of the SU_2 sphere). Suppose $f(x)$ returns either 0 or 1, given any particular n-bit input; the goal is to determine whether or not there exists an x for which $f(x)$ returns 1. For $n + 1$ qubits, the process is as follows: rotate the first n qubits from the ground state into the state

$$\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i,0\rangle,$$

using single-qubit $\pi/2$ rotations, leaving the $(n + 1)^{st}$ bit in the state 0. Then, apply an oracle f to this superposition state, storing the result in the $(n + 1)^{st}$ qubit as $f(i)$. Then rotate the first n qubits back with $-\pi/2$ rotations. Some reasoning about the distribution of pure state amplitudes shows that measuring the first n qubits will give the ground state with probability at least 1/4, and that if this occurs, the system will end up in the state

$$2^n / \sqrt{2^{2n} - 2^{n+1}m + 2m^2} \left|\vec{0}\right\rangle \otimes ((1 - 2^{-n}m)|0\rangle + 2^{-n}m|1\rangle),$$

where there are m solutions to $f(i)=1$. One can then apply the nonlinear operation a polynomial number of times to separate these states until they are visibly distinguishable: the algorithm will require $O((\pi/\alpha)^2)$ operations, where α is defined above. Since quantum mechanics is linear, however, this result has little bearing on reality, although it operates at an interesting crossroads between nonlinear dynamics, speculation on the reality of quantum mechanics, and computational complexity theory (Wei89).

III.2. Fault-tolerant computation

One important aspect of computation in a traditional system, like a silicon chip, is *regeneration*: through nonlinear gain and digital discrimination, one can amplify and correct signals continuously. This is impossible on a quantum computer, where one has a continuous variable which cannot be identically cloned, according to the axioms of quantum mechanics. Nevertheless, paradigms for fault-tolerant quantum computation have arisen over time, and some of these have developed into fairly sophisticated systems for implementing the error-correction discussed earlier in terms of an active computational system.

What fault-tolerance is necessary for classical computers? von Neumann (Neu52) and Gacs (Gac83) suggested means for stochastic analysis of computation with noisy gates and noisy cellular automata, respectively, but silicon computers usually do not need any error-correction whatsoever (or at least such errors are tolerated by the user). The physics is just good enough that we need not worry about active correction. Kitaev (Kit97) has suggested a 2-D anyon-based system for inherently fault-tolerant hardware, but this is very farfetched from an experimental point of view.

Using error correcting codes is not enough to insure the integrity of a quantum computation: how do you know that your ancillae and error-checking routines don't have errors in them as well? How can you perform quantum operations on information that has been abstrusely encoded in the entanglement of the qubits?

To first order, a t-step quantum computation can tolerate errors of $O(1/t)$ per operation, before the probability of getting the correct answer becomes uselessly low. Shor (Sho96b) showed that with error-correction, one could increase the tolerable error per operation to $O(\log(t)^{-c})$, an exponential improvement over the straightforward method. The essential observation is to use an error-correcting code (e.g., a $[n, k, d]$ code as described in **I.2.ii.**) to encode all the qubits into an error resistant form, then performing computations on the encoded qubits, and finally reading out the results. In order to accomplish the latter two steps, one must alternate specially transformed gates (which have been modified to operate on the encoded states, using Clifford group operators and a complicated version of the Toffoli gate, or by using group-theoretic aspects of code stabilizers) with error correction steps (which correct the amplitudes and phases). Excellent speculations on how these operations might be implemented are to be found in (Sho96b) (Got97), and certainly the result that fault-tolerant quantum computation is even theoretically possible provides fascinating insight into quantum mechanics. It is not clear, however, given the many required extra qubits, and the many extra computational steps required to implement fault-tolerant quantum computation, that this will be experimentally useful anytime in the near future.

Finally, we note that if we know the nature of the errors that need to be corrected – for example, in an NMR system we might predict that dephasing due to magnetic field inhomogeneity or inaccurate π pulses might be the dominant causes of error – we can devise specific remedies that might be more effective and less wasteful of qubits than the methods described above. In (Tyc85) and (Fre79) are descriptions of pulse sequence design principles that cause such errors to vanish, by taking advantage of the symmetry properties of the Liouville space of operators, and using iterative schemes to cause successive errors to cancel one another.

"Mein Gott! I'm full of spins."
– E (upon seeing an MRI of his head)

IV. Nuclear Magnetic Resonance

IV.1. NMR Theory

IV.1.i. Noninteracting spins

Nuclear magnetic resonance has proven, since its discovery in the mid-1940s, to be one of the most useful methods for substance characterization and for probing fundamental aspects of chemical and physical behavior. Many models, ranging from the classical to the abstrusely quantum-mechanical, exist for interpreting the phenomenon. The Bloch equations

$$\frac{d\vec{M}}{dt} = \gamma(\vec{M} \times \vec{B}) - \frac{\vec{M}}{T_2} + \frac{M_0}{T_1}\hat{z}$$

describe the time evolution of a classical bulk magnetization $\vec{M}$ under a magnetic field $\vec{B}$. T_1 is the spin-lattice relaxation constant, and describes the rate at which the bulk magnetization aligns with a bias field B aligned along the z-axis; T_2 is the spin-spin relaxation constant, which describes the rate at which any transverse magnetization in the xy-plane decays, due to individual spins dephasing with respect to one another.

A single spin-1/2 particle can be imagined as a simple magnetic dipole in a magnetic field, with Hamiltonian

$$H = -\vec{\mu} \cdot \vec{B},$$

where $\vec{\mu}$, the magnetic moment, can be expressed in terms of the spin operator I as $\gamma\hbar I$. The quantity γ is called the gyromagnetic ratio, and is a characteristic frequency of a nuclear species; almost without exception, it must be determined from experiment. The gyromagnetic ratio determines the size of the nuclear Zeeman energy splittings; if we take the magnetic field $\vec{B}$ to be equal to $B\hat{z}$, then the Hamiltonian for a single spin becomes

$$H = -\gamma\hbar B I_z,$$

and the allowed energy levels are $E = -\gamma\hbar Bm$, where m can take any of the $2I + 1$ values $-I, -I+1, \ldots, +I$. The energy difference between adjacent energy levels, $\Delta E = \gamma\hbar B$, corresponds to the Larmor frequency γB. If one applies radiofrequency energy at the Larmor frequency with a certain phase and for controlled periods of time, one can effect controlled changes in the state of a spin. This observation is key to implementing single-qubit phase shifts, which comprise a significant portion of our vocabulary of elementary gates.

For a proton at room temperature in a 1 Tesla (= 10,000 Gauss) field, about 50,000 times the magnetic field of the Earth, the difference between the population of the ground state and the population of the excited state is about a part in a million; for less sensitive nuclei, it is much smaller. To see this, note that the density matrix of a thermal ensemble,

$$\rho = e^{-\beta H}/Z,$$

can be expanded in a power series in the argument βH, using the definition of the operator exponential (Art). If the Zeeman energy of a spin is significantly less than kT, as it is at room temperature, we can ignore all but the linear term in this power series:

$$\rho = \frac{1}{Z}(1 + \frac{\hbar\gamma B}{kT} I_{kz} + ...)$$

Summing over the states in the Zeeman manifold, one finds that $Z \sim 2I + 1$. If this magnetization is tipped into the transverse plane so that it is visible, one sees a precessing bulk magnetization $M = \chi B/\mu_0$. Using the above formula, one can calculate a *nuclear susceptibility* that follows a version of Curie's law, for spins at finite temperature (Sli90):

$$\chi = \mu_0 M/B = \frac{\mu_0 n \gamma^2 \hbar^2 I(I+1)}{3kT}$$

where n is the number of nuclei per unit volume, and $M = |\vec{M}|$ is the amplitude of the bulk sample magnetization. This equation is particularly useful for evaluating the strength of signals as seen in real NMR experiments at room temperature, since it relates the amplitude of the observed magnetization, M, to the strength of the applied magnetic field, B, and to fundamental parameters of the experiment like the temperature and the gyromagnetic ratio.

In the classical picture, we simply consider the magnetic field B as applying a torque to the spin, causing a change in its angular momentum $J = \hbar I$ according to the equation

$$\frac{d\vec{J}}{dt} = \vec{\mu} \times \vec{B}$$

where we may substitute $\vec{\mu} = \gamma \vec{J}$ in order to get an equation solely in terms of the magnetic moment. This is the first term of the Bloch equation – that is, the Hamiltonian dynamics of a magnet in a field, ignoring T_2 and T_1 relaxation. From the geometry of classical mechanics we know that the equation of motion for a vector $\vec{J}$ rotating with constant angular velocity $\vec{\Omega}$ can be written as

$$\frac{d\vec{J}}{dt} = \vec{\Omega} \times \vec{J}$$

Therefore we may consider a *rotating frame* in which the coordinate system rotates at velocity $\vec{\Omega}$, and combine the above two classical equations to derive the equation of motion of a spin in this rotating frame,

$$\frac{d\vec{\mu}}{dt} = \vec{\mu} \times \left(\gamma \vec{B} + \vec{\Omega}\right).$$

The term in parentheses can be thought of as an effective magnetic field. When Ω is precisely $-\gamma B$, the effect of the magnetic field vanishes in the rotating frame, and the spin appears stationary in this frame; this is the resonance condition. This has profound consequences for an RF (radiofrequency) magnetic field, say $\vec{B}_1 = B_1 \cos(\omega t)\,\hat{x}$, which can be written as

$$\vec{B}_1 = \frac{B_1}{2}(e^{i\omega t} + e^{-i\omega t})$$

If we judiciously choose a rotating frame rotating at ω, then the RF field looks like

$$\vec{B}_1' = \frac{B_1}{2}(1 + e^{-i2\omega t}).$$

Note that the component which was originally moving at $+\omega$ now appears stationary to an observer in the rotating frame, while the component at $-\omega$ now seems to be rotating twice as fast (and can therefore be neglected). Thus in the rotating frame at resonance, the sample magnetization rotates about the apparently stationary magnetization $\vec{B}_1'$ at the rate of γB_1 radians per second. The result is that we have very precise spin control, and can use RF pulses to change the state of a spin to virtually any state we wish. For example, we may transform longitudinal magnetization into transverse magnetization simply by applying a short pulse for which $\gamma B_1 = \pi/2$; we will call this a $\pi/2$ pulse. In our earlier notation this corresponds to either an $X(\pi/2)$ or a $Y(\pi/2)$ pulse, for example. Thinking about the rotating frame in three dimensions will persuade you that the only difference between an $X(\pi/2)$ pulse and a $Y(\pi/2)$ pulse is that the phase of the $Y(\pi/2)$ sinusoidal oscillation starts $\pi/2$ radians beyond the phase of the $X(\pi/2)$ pulse, so that our vector $\vec{B}_1$ is aligned along the *y*-axis in the rotating frame, as opposed to being aligned along the *x*-axis in the rotating frame. After being tipped into the *xy*-plane, the spin rotates at the Larmor frequency.

Spin-spin and spin-lattice interactions cause the population of spins to decay to their thermal Boltzmann distribution. As a result of the first effect, spin-spin relaxation, the transverse magnetization of a spin after a single $\pi/2$ pulse exhibits a sinusoidal signal with a decay envelope proportional to e^{-t/T_2}. This is called a *free induction decay* (FID), and is due to the dephasing of spins throughout the sample because of environmental and quantum-mechanical reasons, including the presence of bias magnetic field inhomogeneities in the sample, local intermolecular and intramolecular magnetic interactions, and the diffusion of molecules to areas with different magnetic fields: the result of these magnetic effects is that spins radiate energy at slightly different frequencies, and as the phases begin to appear random with respect to one another, their sum converges to zero. A π pulse, on the other hand, rotates a spin initially aligned with the *z*-axis by 180^o, from the up into the down position, and thus no transverse signal appears. The spin relaxes back to its upward position with a time constant T_1; this process is called spin-lattice relaxation. It results from interactions with a spin and a thermal bath of magnetic dipoles, an electromagnetic field, or any other quantum system with which the spins might become entangled through interactions. T_1 and T_2 effects are explicitly included in the Bloch equations which appeared at the beginning of this section. It is simple to see that spontaneous relaxation (due to coupling to the vacuum modes) has an insignificant influence upon nuclear magnetic relaxation processes. By considering the density of radiative modes in a vacuum of volume *V*, it follows that the energy density goes as the frequency cubed,

$$D(\upsilon) = \frac{8\pi V \nu^2}{c^3} \rightarrow E(\upsilon) \propto \nu^3$$

and so at RF frequencies (in our case, 10-50 MHz), the relaxation through vacuum mode coupling is almost completely negligible, since the magnetic field energy density of the vacuum modes is so small. It follows that stimulated emission (exposure to energy at the Larmor frequency), and random interactions (which cause spins to become entangled with other systems) are the only T_1 pathways which a nucleus might take in order to relax.

Some specific pathways have been evaluated as likely contributors to the fluctuating magnetic field responsible for spin-lattice relaxation, including dipole-dipole interactions, quadrupole

relaxations, spin rotation, and scalar relaxation. In dipole-dipole coupling, the dipoles interact via $\mu_1\mu_2/r^3$ terms in the spin Hamiltonian; a moving dipole creates randomly varying magnetic fields as it tumbles by. Introduction of even a few paramagnetic ions into the system causes strong dipole-dipole couplings – indeed, even the presence of dissolved oxygen from the air is known to decrease T_1 by a factor of 2 in water at 300 K. Quadrupole relaxation occurs when an asymmetric charge distribution interacts with an electric field gradient; if a nucleus with a quadrupole moment rotates or vibrates in such an environment, this effect can be significant. Ionic structures, polarized bonds, and crystals all possess significant electric field gradients at the nucleus; but since ^{1}H, for example, possesses no quadrupole moment, it is insensitive to its electrical environment. Spin rotation results when nuclei and electrons create currents via their random tumbling, which equate to creating randomly varying magnetic fields. Spin rotation is a very important relaxation pathway for small molecules like water. Scalar relaxation occurs when neigboring spins in a single molecule interact, or 'cross-polarize,' and is a fairly subtle effect, since it requires the creation of higher quantum coherences. Most lattice energies, like those of translation and rotation modes, fall into the low IR range and thus are not particularly effective at stimulating relaxation at frequencies as low as 10-50 MHz. Indeed, the T_1 of a nuclear spin can be as long as several hours for a cool sample of pure water; few quantum systems demonstrate such robustness in ordinary environments. The nucleus is a well-isolated and highly stable environment.

In the quantum picture, the time-evolution operators for the Hamiltonian $H = -\gamma\hbar B I_z$ cause the Heisenberg time-evolution equations to become

$$\frac{dI_{x,y}}{dt} = \frac{i}{\hbar}\left[H, I_{x,y}\right] = \pm\gamma B I_{y,x} \text{ and } \frac{dI_z}{dt} = 0$$

which, upon taking the expectation value, and taking $\vec{\mu} = \gamma\hbar I$, instantly gives the Ehrenfest equation

$$\frac{d\langle\vec{\mu}\rangle}{dt} = \langle\mu\rangle \times \gamma\vec{B}$$

which is identical to the classical picture, as expected. The Hamiltonian corresponding to a bias field $\vec{B}_0 = B_0\hat{z}$ plus a rotating RF field $\vec{B}_1 = B_1\cos(\omega t)\,\hat{x}$,

$$H(t) = -\gamma\hbar\left[B_0 I_z + B_1\left(I_x\cos(\omega t) + I_y\sin(\omega t)\right)\right] = -\gamma\hbar[B_0 I_z + B_1 e^{-i\omega t I_z} I_x e^{i\omega t I_z}]$$

suggests that we create the analogue of the rotating frame by redefining $\Psi = e^{-i\omega t I_z}\Psi'$. Taking Ψ' and substituting it into the Schrodinger equation, we get

$$\frac{\hbar}{i}\frac{d\Psi'}{dt} = [\hbar(\omega + \gamma B_0)I_z + \gamma\hbar B_1 I_x]\Psi'$$

which is the quantum-mechanical analogue to the rotating frame equation; the time-dependence of the Hamiltonian has been removed by working in the *interaction picture*, and the equation of motion is now that of a spin in a static field. Using the time-evolution equation,

$$\Psi'(t) = e^{-iHt/\hbar}\Psi'(0),$$

and converting back into the lab frame gives the formal solution to the quantum mechanical problem,

$$\Psi(t) = e^{-i\omega t I_z} e^{-i((\omega+\gamma B_0)I_z + \gamma B_1 I_x)}\Psi(0)$$

Calculating the expected magnetizations in the lab frame proceeds directly by considering the

commutators of I_x and I_z, which are given in **I.1.i.**. For example, $\langle \Psi^* | \mu_x | \Psi \rangle$ immediately gives the expected value

$$\langle \mu_x(t) \rangle = \langle \mu_x(0) \rangle \cos(\gamma B_1 t),$$

and indeed the entire classical picture applies, justifying the above 'vector' model of pulsed NMR for a single noninteracting spin.

Before adding interactions to the picture, we consider some simple phenomena of single-spin NMR. For example, if a $\pi/2$ pulse is applied to a sample at time $t = 0$, a free-induction decay signal appears. If a π pulse is then applied at time $t = \tau$, the signal will increase until it reappears fully at time 2τ, with an *exponentially increasing* envelope from τ to 2τ. This is called a spin echo, and is depicted in **Figure 1**.

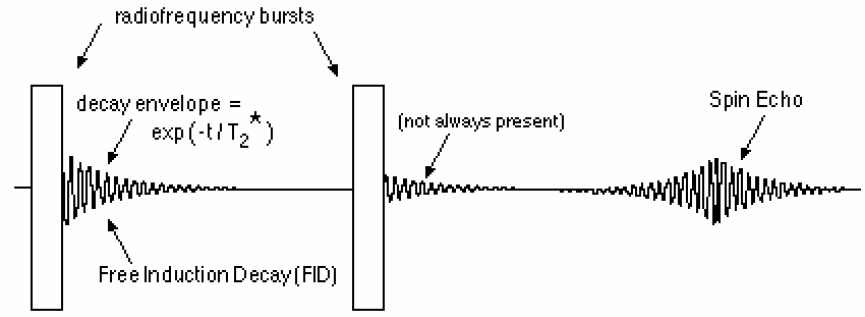


Figure 1: Spin-echo (Kir95)

If one works through this process using the classical picture described at the beginning of this section, this is not surprising; the spin-spin relaxation that leads to the T_2 decay is a result of the spins' magnetizations becoming out of phase with one another, and is a completely reversible process (up to the order of diffusion effects and thermal relaxation), since the information needed to restore the spins to their original state is encoded in the local magnetic field. It is easy to understand via the following vector-diagram, which is a common tool in elementary pulse NMR:

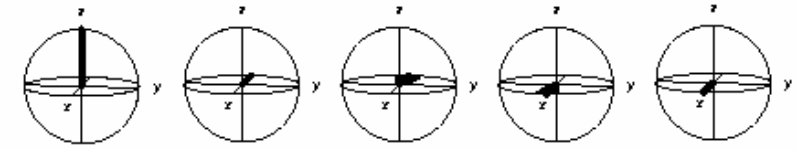


Figure 2: The vector picture of the spin-echo (Ger97a)

In the first frame of **Figure 2** we have a bulk magnetization pointing upwards. We apply a 90^o pulse about the *y* axis, immediately tipping the spins over to get the state in the second frame. The magnetization precesses for a time τ, bringing us to the third frame (where the spin-spin dephasing is depicted by a small deviation in the various magnetization vectors). Now we apply a 180^o pulse about the *y* axis, which flips the spins (bringing us to frame four). Now all of the spins are precessing in a direction *opposite* to the one they were just heading in; in particular, spins that were diverging at a certain rate in the first part of the spin-echo experiment are now moving just as quickly in the opposite direction in the second part. At a time 2τ we see a full signal again (as shown in the final frame), since the spins are now all in phase with one another again, and their signals are again mutually coherent.

IV.1.ii. Interacting spins

So far we have considered the interactions of a magnetic field with single spins. All of the computational effects and analytical boons of NMR, however, are due to the interactions of spins with each other. This makes NMR incredibly complex; for our purposes, we will only consider *scalar* couplings between different spin-1/2 particles on the same molecule. These couplings, often called J-couplings, don't depend on the absolute orientation of the molecule with respect to the bias magnetic field. The Hamiltonian of a spin system with scalar couplings is given by

$$H = \sum_{i=1}^{n} -\gamma_i \hbar B_0 I_{iz} + \sum_{i<j} 2\pi J_{ij} \hbar I_{iz} I_{jz}$$

where J_{ij} describes the coupling, in Hertz, between spins *i* and *j*. The form of this coupling is somewhat abstruse – why should it be proportional only to the *z*-component of each spin? The answer is that actual Hamiltonian term corresponding to the interaction between spins *i* and *j* is of course a complicated dipole-dipole coupling term, which in classical electrodynamics is described by

$$\frac{\vec{\mu}_i \cdot \vec{\mu}_j}{r^3} - \frac{3(\vec{\mu}_i \cdot \vec{r})(\vec{\mu}_j \cdot \vec{r})}{r^5},$$

where $r = |\vec{r}|$ is the distance between the nuclei. In the high magnetic field $B\hat{z}$ limit, this becomes

$$\frac{\gamma_i \gamma_j \hbar^2 (1 - 3\cos^2\theta)}{4r^3}(3I_{iz}I_{jz} - I_i \cdot I_j)$$

where θ is the angle between $\vec{r}$ and the *z*-axis; we can ignore other energy terms that correspond to processes or frequencies that occur far from the Larmor resonance condition for flipping each spin. To simplify this further, one can invoke arguments based on the properties of a room-temperature liquid ensemble in the high-field limit, using the *method of moments*, to conclude that the coupling can be written simply as

$$\frac{\gamma_i \gamma_j \hbar^2 (1 - 3\cos^2\theta)}{r^3} I_{iz} I_{jz},$$

which is simple enough make tractable the prediction of the time-evolution of a spin system. The logical flow behind this series of simplifications is well-explained in (Sli90). From now on, we will take the scalar J-coupling model to be factual, and will not delve into the physics below this level, although it is quite possible that in the future, when unconventional cooling techniques, far weaker or far stronger magnetic fields, or long pulse sequences become commonplace, the J-coupling model will cease to be sufficient for predicting the outcome of an NMR pulse sequence.

As noted in **I.1.i.**, if we are given a time-evolution operator

$$U(t_1, t_0) = e^{iH(t_1 - t_0)/\hbar}$$

where *H* is the Hamiltonian of the quantum system during the period from time t_0 to time t_1, then the density matrix evolves as

$$\rho(t_t) = U\rho(t_0)U^{-1}.$$

We often begin experiments at $t_0 = 0$ with the thermal density matrix

$$\rho(0) = \frac{1}{Z}(1 + \sum_{i=1}^{n} \frac{\hbar \gamma_i B}{kT} I_{iz} + ...).$$

The only time-evolution operators of interest will be the single-qubit rotations ($\theta \equiv \gamma B_1 t$),

$$X_i(\theta) = e^{i\gamma B_1 t I_{ix}}$$
$$Y_i(\theta) = e^{i\gamma B_1 t I_{iy}}$$
$$\text{and } Z_i(\theta) = e^{i\gamma B_1 t I_{iz}}$$

(the last of which shares its functional form with the free evolution of a spin in the biasing magnetic field), and the J-coupling operator between two spins,

$$ZZ_{ij}(\theta) = e^{-i2\pi J_{ij} t I_{iz} I_{jz}}.$$

Since the single-qubit rotations result from strong, short pulses of radiofrequency energy, we often assume them to take no time at all, which lets us ignore the concurrent free evolution as insignificant.

The key observation which allows us to simplify the computation of the time-evolution of a multiparticle spin system is that we can express *the density matrix itself* in a Liouville-space basis consisting of operators. At first glance, this is not surprising, since the density matrix is indeed an operator in the 2^{2n}-dimensional Liouville space. The basis that we choose is the basis of operators

$$2^{k-1}\prod_{i=1}^{n} I_{ia_i}^{b_i}$$

where b_i is 1 for k of the n terms in the product and 0 for the other $n - k$ terms, and a_i is one of x, y, and z for each i. This basis is complete, since it contains precisely 4^n terms (for each i between 1 and n, a term of the product can take one of four possibilities – it can be x, y, z, or absent), and all the resultant matrices are linearly independent (Ern90).

For example, consider the following deviation density matrix for two spins, labeled 1 and 2, with respective chemical shifts ω_1 and ω_2 in a particular rotating frame:

$$\rho = I_{1z} + I_{2z},$$

This density matrix could be, for example, an unnormalized thermal density matrix with the unobservable identity term subtracted off. Let's say we apply a pulse which rotates spin 1 90° about the x-axis, and then let the system evolve for $\tau = 1/J_{12}$ seconds. The result of the first operation is a density matrix in the state

$$X_1(\pi/2)(I_{1z} + I_{2z})X_1(-\pi/2) = e^{i\pi/2 I_{1x}} I_{1z} e^{-i\pi/2 I_{1x}} + I_{2z},$$

where we have noted that all operators which purely refer to spin i commute with those that refer solely to spin j (in particular, rotating 1 has no immediate effect on spin 2). For a spin-1/2 particle, for which $I_x^2 = I_y^2 = I_z^2 = 1/4$, we can miraculously expand the operator $X_1(\theta)$ as

$$\begin{aligned} X_i(\theta) &= e^{i\theta I_{ix}} \\ &= 1 + i\theta I_{ix} + (i\theta I_{ix})^2/2! + (i\theta I_{ix})^3/3! + \ldots \\ &= 1 + i\theta I_{ix} - (\theta/2)^2/2! - 2iI_{ix}(\theta/2)^3/3! + \ldots \\ &= \cos(\theta/2) + 2iI_{ix}\sin(\theta/2) \end{aligned}$$

and now it is trivial to evaluate the result of the first pulse, simply by using the commutator equations for the spin operators given in **I.1.i.** The result is

$$I_{1y} + I_{2z}$$

as one might expect from the single-spin vector picture given in the last section. Now for the time-evolution, which lasts a duration $1/J_{12}$: each spin undergoes a free evolution for $1/J_{12}$

seconds, along with a J-coupling interaction during the same period of time. The final state is

$$Z_1(\omega_1\tau)Z_2(\omega_2\tau)\ ZZ_{12}(2\pi)(I_{1y} + I_{2z})ZZ_{12}(-2\pi)Z_2(-\omega_2\tau)Z_1(-\omega_1\tau).$$

One might wonder in what order to apply the three time-evolution operators, but as it turns out they all commute, so it doesn't matter. The following table, which is derived using the same 'miracle' we used in the derivation of $X_i(\theta)$, is useful for evaluating such products (Sli90):

$U\downarrow \qquad I_{1a}\rightarrow$	I_{1x}	I_{1y}	I_{1z}
$X(\theta)$	I_{1x}	$I_{1y}\cos(\theta) - I_{1z}\sin(\theta)$	$I_{1z}\cos(\theta) + I_{1y}\sin(\theta)$
$Y(\theta)$	$I_{1x}\cos(\theta) + I_{1z}\sin(\theta)$	I_{1y}	$I_{1z}\cos(\theta) - I_{1x}\sin(\theta)$
$Z(\theta)$	$I_{1x}\cos(\theta) - I_{1y}\sin(\theta)$	$I_{1y}\cos(\theta) + I_{1x}\sin(\theta)$	I_{1z}
$ZZ(\theta)$	$I_{1x}\cos(\theta) + 2I_{1y}I_{2z}\sin(\theta/2)$	$I_{1y}\cos(\theta) - 2I_{1x}I_{2z}\sin(\theta/2)$	I_{1z}

Using this table it is trivial to see that the end result of time-evolution for $1/J_{12}$ seconds is

$$-I_{1y}\cos(\omega_1\tau) - I_{1x}\sin(\omega_1\tau) + I_{2z}$$

The observable magnetization at time $t = \tau$ corresponds to $\mathrm{Tr}(I^+\rho)$; to evaluate this, the matrix representation of the SU_2 group, given in **I.1.i.**, might be useful.

We briefly list some pointers to further topics in NMR theory, an incredibly vast and beautiful subject. In (Ern90) are presented many pictures of NMR, including many different flavors of Liouville operator spaces and superoperator spaces; Ernst (who won the Nobel Prize for his work in NMR) also has cast quantum computing explicitly in terms of Liouville space, a useful tool for interpreting the Tycko-Guckenheimer-Pines iterated pulses which correct their own errors, for example (Ern98) (Tyc85). Ernst also delves into many interesting aspects of NMR theory, including stochastic-reponse aspects of spin systems, methods of recursively expanding pulse sequences, aspects of multiple-quantum coherences (an *n*-quantum coherence decays at the exaggerated exponential rate e^{-nt/T_2} for example), and methods of understanding multiplets in spectra of molecules with complex interactions.

IV.1.iii. Spin phenomenology

In a recent paper (Lloy), the nature of the spin-echo is discussed in terms of algorithmic complexity theory. During the period between the start of a spin-echo experiment and the time *t* when a π pulse is applied, the entropy of the spins increases; the entropy of the spins then decreases until the time 2*t* when the spins have returned to their original state. Since the phase of a spin with respect to the other spins in the ensemble can be recreated from knowledge of the local Larmor frequency and the elapsed time since the start of the experiment, and the local Larmor frequency is encoded in the local magnetic field (assumed to be a constant over the course of the experiment), the only information which an observer must possess in order to observe the spin-echo signal is how long a time *t* to wait after the π pulse occurred: the algorithmic complexity thus is proportional to log(*t*), since that's how many bits it takes to store the numerical value of the waiting time (up to any chosen coarse-graining of time). The reason that this value is important is because observing a spin-echo could be used to extract energy or information from the system, and so we must consider all possible costs of being able to extract this energy, in order to properly judge the entropy of the system.

The experiment proceeds as follows: as the spins become correlated to the magnetic field, the entropy of the spins increases, and the mutual information between the spins and the field increases; as the spins decorrelate from the magnetic field in the second part of the spin-echo experiment, the entropy of the spins and the mutual information between the spins and the field both decrease to their original values, requiring only the log(*t*) bits of information that an observer needs in order to see the spin-echo. Lloyd and Zurek present a QED justification, using radiofrequency fields in a coherent state (e.g.,

$$\left|\psi_{rf}\right\rangle = e^{-a^2/2}\sum_{n=0}^{\infty}(a^n/n!)\left|n\right\rangle,$$

a good model for a real-life macroscopic RF field, since subtracting a photon from the field leaves the state unchanged: the coherent state is an eigenstate of the lowering operator) to show that the spins do not contribute entropy to the RF field during the second half of the spin echo experiment, and that the final spin state is indeed a pure state at time 2*t*, as expected. The total entropy (fine-grained, and evaluated according to traditional statistical mechanics) is constant, just as one would expect for a non-dissipative system, such as one describable by a Hamiltonian.

The moral of the story is that "a device must have an algorithmically exact description of the form in which energy is distributed" in a system under observation, and that "since the algorithmic complexity of Hamiltonian systems increases as log(*t*) over time, more energy is required to get work out of such systems, even though the fine-grained statistical entropy remains constant" (Lloy).

Using NMR, a quantum-mechanical Maxwell's demon (QMMD) could be constructed (Llo96); for example, whenever a spin in a thermal ensemble is in an excited state, a π pulse could be used to extract its energy, and convert heat into work. Of course, Landauer's result, which says that erasing a bit increases the entropy of the environment by at least $S_l = k\ln 2$, shows that the Second Law is not violated; rather, the QMMD operates as a heat engine. Since quantum measurement generates more information (via collapse of a possibly intricate wavefunction) than classical measurement, a quantum system is even more inefficient at extracting energy than a classical one. For example, suppose a measurement

$$\rho \to \rho'$$

takes place, where ρ' is a pure state if the measurement is with respect to some orthogonal basis, as is often the case; the information generated is equal to the change in the von Neumann entropy,

$$\Delta S = -Tr\rho'\log\rho' + Tr\rho\log\rho = Tr\rho\log\rho$$

and thus *intrinsically quantum information* is generated, which is superfluous and possibly unrelated to the classical information that we extracted from the system. A quantum heat engine must waste more than a classical one (Llo96). The CNOT operation can be used to extract information from a spin as follows: perform $\mathrm{CNOT}_{1,2}$ (which flips 2 if 1 is excited), then $\mathrm{CNOT}_{2,1}$ (which flips 1 if 2 is excited); if the dipole moment of 1 is bigger than that of 2, energy $2(\mu_1-\mu_2)$B is transferred to the field from the spins; if the spins are biased by a magnetic field and exposed to separate thermal reservoirs, letting them reach thermal equilibrium will result in the completion of a Carnot cycle. But the cost is great: when the spins are in contact with their respective reservoirs, they *decohere* (and are 'measured') by T_1 processes, and lose their phases (or mutual information), while becoming entangled with their environment.

Certainly a method of predicting the density matrix's entropy and degree of entanglement, and how it changes with respect to T_2 and T_1 processes over time, is crucial for understanding how one should arrange information in a thermal ensemble so as to make quantum computation efficient and long-lived. In part **V.1.iii.** we will discuss what practical implications might result from this observation, so as to make quantum computation and materials characterization practical on real molecules and polymers.

IV.2. Bulk Spin Resonance Quantum Computation

Several proposals for bulk NMR computation have been suggested, including logical labeling, spatial averaging, and temporal averaging; all three methods have been experimentally realized. In general, the theme is that a set of operations is applied to a large ensemble of spins which comprises part of an effective pure state, resulting in a step of a computation. The bulk state is then read out, thus measuring the expectation value of a certain observable over the ensemble. In practical implementations, often several experiments must be run probabilistically (logical labeling), sequentially (temporal averaging), or in parallel (spatial averaging) – the results are then selectively processed, mathematically added together, or physically added together, respectively, so as to effectively synthesize a computation on a pure state.

A thermal ensemble of spins in a bias magnetic field has a density matrix

$$\frac{1}{Z}(1+\sum_{i=1}^{n}\frac{\hbar\gamma_i B}{kT}I_{iz}+...)$$

which can be separated into two parts, a homogeneous background component ($1/Z$) due to the high temperature of the sample, and a perturbation component, called the deviation density matrix,

$$\rho_\Delta = \rho - \mathrm{Tr}(\rho)/2^n = 1/Z \sum_{i=1}^{n}\frac{\hbar\gamma_i B}{kT}I_{iz}$$

whose magnitude is roughly 10^6 times smaller than the background component, and which is a manifestation of the spin polarization due to the magnetic field. This perturbation component is the basis for all the manipulations used in nuclear magnetic resonance, and we will completely neglect the rest of the density matrix since it is never observable. From this deviation density matrix we can extract an *effective pure state* (e.g., a state which to an observer looks pure) for quantum computation. One nice image is that of a 'soloist' standing out from a 'choir' of background spins (Chu98a) (Ger97a).

One can use logical labeling to generate an effective pure state from a deviation density matrix by using certain spins as labels (Chu98a). Consider preparing the density matrix ρ (neglecting the homogeneous background component, of course) with an operator P, as

$$P(\rho) = \sum_k |k\rangle\langle k| \otimes \rho_k .$$

where $|k\rangle\langle k|$, $k = 0,1, \ldots$ are the possible states of a particular ancilla spin (or register of spins), whose state, when measured, indicates the contents of the rest of the density matrix. We can

prepare ρ_0 as an effective pure state, $|\phi_0\rangle\langle\phi_0|$, and treat all the other ρ_k as junk states. If we define a conditional readout step R as

$$R(P(\rho)) = (|0\rangle\langle 0| \otimes I)P(\rho)(|0\rangle\langle 0| \otimes I),$$

then one way to implement a computation C so that it operates upon an effective pure state is

$$C'(\rho) = R((I_{anc} \otimes C)(P(\rho))) = |0\rangle\langle 0| \otimes C|\phi_0\rangle\langle\phi_0|,$$

where I_{anc} is the identity over the ancilla (label) states, since the true computation C operates only upon the ρ_k. Therefore if we condition on the ancilla spins, and accept the answer only if the ancilla bit is in the 0 state, we get an effective pure state. Of course, if we observe that our ancilla bit is in a state other than 0, we must discard the experiment, since it is therefore the result of a computation performed on one of the junk states. This situation may seem contrived, but it actually occurs very naturally when CNOTs are used to redistribute quantum populations – after each CNOT, the density matrix is very naturally left in a conditional state. A specific algorithm for accomplishing such a redistribution with logical labeling is given in (Chu98a).

Spatial labels can be devised by applying unitary preparation operators, such as gradient RF pulses, which prepare spins in a manner dependent upon their position within a bulk sample (Cor97). The preparation operators are designed so that the transformed density matrix, which one then uses for computation, has the properties of an effective pure state (as discussed above) – the only difference is that the various experiments occur in parallel, in different parts of the sample, rather than in series or in a probabilistic manner. Due to the hardware complexity of implementing precise gradient pulses, we do not discuss this result further at this point.

In temporal labeling, one runs a series of experiments which start with different unitary preparation operators, and then averages the experimental results so that the resultant ρ appears to have been in a pure state. Temporal averaging is a very practical proposal for a small number of qubits, since no qubit has to be reserved for conditioning (as was required for logical labeling) (Kni97): the most basic and obvious method is *exhasutive averaging*: if one runs 3 experiments on $^{13}CHCl_3$, for example, starting with deviation density matrices

$$\begin{bmatrix} 1 & & & \\ & .6 & & \\ & & -.6 & \\ & & & -1 \end{bmatrix}, \begin{bmatrix} 1 & & & \\ & -.6 & & \\ & & -1 & \\ & & & .6 \end{bmatrix}, \text{ and } \begin{bmatrix} 1 & & & \\ & -1 & & \\ & & .6 & \\ & & & -.6 \end{bmatrix},$$

then averaging the 3 experimental results gives the same result as if one had performed an experiment starting with a state equal to the average of the 3 initial density matrices,

$$\begin{bmatrix} 1 & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{bmatrix},$$

which happens to be a pure state. In general, for n qubits one must perform 2^n-1 experiments, applying each permutation in the cyclic group C_{2^n-1} so as to eliminate the 2^n-1 useless state amplitudes in the density matrix. The signal-to-noise ratios (SNRs) that result from temporal averaging are fairly good due to the high number of experiments, as long as errors are not

induced by spectrometer drift or other nonidealities. However, the exponential investment required to perform temporal averaging can be prohibitive for large numbers of qubits. A more frugal method is the algorithm called *flip-and-swap* in (Kni97): since for any state *w*, the density matrix entries for the populations in *w* and $\overline{w}$ (the state with the spins of *w* flipped) satisfy

$$\rho_{ww} + \rho_{\overline{ww}} = 1/2^{n-1},$$

one can average the results of two experiments, the first with no preparation, and the second prepared by first inverting all the qubits and then switching the states $|0\rangle$ and $|2^n - 1\rangle$ (e.g., the ground and highest-excited states); the resultant signal is as if the experiment were performed on the initial state

$$\alpha/2^n (|0\rangle\langle 0| - |2^n - 1\rangle\langle 2^n - 1|),$$

where α is the average gyromagnetic ratio of all the spins, multiplied by nB/kT. The polarization in the all-1 state can be dispersed by mapping this polarization to a randomly chosen state after the flip-swap operation, and then averaging. If one has an ancilla qubit, for a total of $n + 1$ qubits, one can use it as a logical label, with a little bit of pre- and postprocessing: the idea is that if the ancilla is 0, the other *n* qubits are in an effective pure state with 'soloist' $|0\rangle$; if 1, then the other *n* qubits are in a state with soloist $|2^n - 1\rangle$. The pre- and postprocessing steps are given in (Kni97), where it is suggested that labelled flip-and-swap is one of the most efficient preparations in terms of expected SNR and minimal waste of qubits.

Another temporal averaging strategy is to randomize over groups of unitary operators, by choosing preparation operators *P* from some distribution on a group {*P*}; the minimum requirement for a procedure of this sort is that $\langle P\rho P^{-1}\rangle_P$ be an effective pure state. Such strategies effectively result in a state of the form

$$(\rho_{00} - \bar{p})\ |0\rangle\langle 0| + \bar{p}\, I,$$

so that the expectation value of a measurement σ_x after a computation *C* is

$$\bar{r} = \mathrm{Tr}(CP\rho P^{-1}C^{-1}\sigma_x) = \mathrm{Tr}(P\rho P^{-1}C^{-1}\sigma_x C) = (\rho_{00} - \bar{p})\mathrm{Tr}(C|0\rangle\langle 0|C^{-1}\sigma_x),$$

which is proportional to the result expected of a pure state computation. Due to the statistical nature of this form of temporal averaging, the number of experiments required to determine $\bar{r}$ to within a fraction ε with probability *c* is $-\log(c)/(\varepsilon^2 \mathrm{SNR}^2)$.

Several historical experiments with conventional NMR spectrometers have performed, including state tomography (Chu98a), error correction (Cor98), and a Grover's search on 2 qubits (Chu98b) (Jon98b). Experiments have also been performed on *quantum error correction* (Cor98), in which it was found to be possible to correct phase errors due to T_2 effects on a liquid NMR sample. (We briefly mentioned this experiment in **I.2.ii.**) A state $|a_1a_2a_3\rangle$ can undergo a random change to

$$e^{-i\vec{\theta}\cdot\vec{\sigma}}|a_1a_2a_3\rangle,$$

for example, where θ is a vector of random phases. According to this error model, to first order a qubit encoded as

$$\alpha|+++\rangle + \beta|---\rangle$$

would be transformed into the state

$$\alpha|+++\rangle + \beta|---\rangle - i\theta_1(\alpha|-++\rangle + \beta|+--\rangle) - i\theta_2(\alpha|+-+\rangle + \beta|-+-\rangle) - i\theta_3(\alpha|++-\rangle + \beta|--+\rangle)$$

Note that we are assuming that the error is common to all three qubits – this works nicely for errors that result from, say, inhomogeneities in the magnetic field. One can diagnose the error without destroying the quantum coherences because the errors map the encoded state onto orthogonal subspaces. One can simulate T_2 effects by setting up a field gradient $\partial B_z / \partial z$ and letting the molecules diffuse for a time τ and distance dz, resulting in a random phase $p\tau\gamma\, dz\, \partial B_z / \partial z$, for a p-quantum coherence (see **IV.1.ii.**). (Note that zero-quantum coherences don't experience T_2 effects at all.) In the (Cor98) experiment, Cory et. al. started with the intial state $I_{1z}(1/2+I_{2z})(1/2+I_{3z})$ (one upwards-pointing spin containing the signal, plus two ground-state ancillae), and applied the encoding pulse sequence

$$\mathrm{CNOT}_{1,2}\,\mathrm{CNOT}_{1,3}\,\mathrm{Y}_1(\pi/2)\,\mathrm{Y}_2(\pi/2)\,\mathrm{Y}_2(\pi/2),$$

giving $(I_{1x} + I_{2x} + I_{3x}+4I_{1x}I_{2x}I_{3x})$, whose first and third-order coherences exponentially decays with time constants T_2, $T_2/3$, respectively. The experimenters then corrected this decay with the inverse encoding sequence, followed by a Toffoli gate that flipped the first spin in proportion to the amplitude of the two ancillae. Their experiment showed that they could reduce the initial decay slope by a factor of 5.

Since polarization in NMR systems is rather low (1 part in a million), the exponential losses associated with signal declines do not yet justify the addition of ancilla qubits, and so for now error-correction is mostly a curiosity, although with the development of advanced polarization techniques (like optical pumping) perhaps this will someday be useful. For n weakly-interacting particles, pulsed-NMR manipulations can improve the polarization of a particular qubit only by a factor of $O(n^{1/2})$; therefore 'computational cooling' cannot offset the exponential decay in signal that one obtains by adding additional qubits. We begin the next section by quantifying the relationship between the NMR signal strength and the number of qubits.

IV.3. Coherence, Signals, and NMR Computing

The maximum possible signal obtainiable from an NMR experiment is approximated by the largest eigenvalue of the density matrix (Chu98a); for the lowest energy state this eigenvalue is

$$\gamma\,\hbar\, Bn/(2^{n+1}kT),$$

giving a transverse magnetization of

$$n\gamma\,\hbar\,(\gamma\,\hbar\, Bn/(2^{n+1}kT)).$$

Thus there is a hidden exponential cost – the signal exponentially decreases with the number of qubits! This is not a condemnation of bulk quantum computing, only of thermal quantum computing, since the procedure of creating the initial polarization through thermal relaxation in a bias field results in most of the state amplitudes being homogenized. Even worse, it has been proven that *no NMR quantum computation experiment* (up to the current day) has ever created any real entanglement, in the sense that all of the resultant density matrices are completely separable (Bra98). Their proof is constructive, in that separations are specified for all density matrices in a neighborhood of the maximum-entropy density matrix, for bulk thermal quantum computations involving fewer than 14 qubits, at room temperature in magnetic fields of O(1 Tesla). (It is conceivable, but not guaranteed, that performing bulk thermal quantum computation on more than 14 qubits could result in entangled states.)

Nevertheless, NMR quantum computation is interesting from the perspective of creating useful computations for the purpose of efficiently interrogating and characterizing natural materials. Therefore we will persist with an analysis of NMR signals at the level of interest of the NMR spectrometer engineer. For conceptual simplicity, we will consider this issue at two levels – that of the physics, and that of the instrumentation. The signal, as noted in **IV.1.i.**, is a voltage in an *N*-turn coil (inductor) of area *A*, volume V_{coil}, and quality Q , which according to Faraday's Law, gives

$$V = L\frac{dI}{dt} = -\mu_0 NAQ\ \frac{dM_x}{dt} = -\frac{d}{dt}\int M_x \beta dV$$

where β is, by the principle of reciprocity, the magnetic field generated by a current of 1 A going through the coil. The Q of a coil is equal to $\omega L/R$, where $L = \beta^2 V_{coil}/\mu_0$ is the inductance, ω is the frequency of interest, and R is the resistance of the coil wire (Hay96). Defining the *fill fraction* $\eta = V_{sample}/V_{coil}$, and substituting in for M_x using the Bloch equations, we can write down the following equation for the NMR signal seen by a long solenoidal coil (Gre98):

$$V = \frac{\chi \eta B}{2}\sqrt{\frac{T_2}{T_1}\left(\frac{\omega V_{coil} QR}{\mu_0}\right)},$$

where χ is the Curie's-law susceptibility defined in **IV.1.i.** In order to compute the total signal-to-noise ratio (SNR), we can compute the Johnson noise of the inductor as (Rei65)

$$V_n = \sqrt{4kTR\Delta f F}$$

where Δf is the bandwidth of the system and $F \equiv 10^{NF}$ is the characteristic noise of the amplifier. F is defined to be the ratio of the input SNR to the output SNR, and *NF* is the *noise figure*, measured in dB and often provided by amplifier manufacturers as a measure of an amplifier's performance. The total SNR is V/V_n, roughly 100,000 for a typical spectrometer at 1 T. (Our spectrometer easily attains a total SNR of 10,000,000 – see **VI.1.i.**)

A more sophisticated model, which takes into account the skin depth of the coil, as well as the fact that different sections of the coil receive signals out of phase from one another, results in the following equation for the SNR (Hou76):

$$\text{SNR} = \frac{\beta \chi V_s}{2.37\gamma\mu_0}\sqrt{\frac{2\pi r}{FkTl\xi\Delta f}}\left(\frac{\omega_L^7}{\mu\mu_0\rho}\right)^{1/4},$$

where r is the radius of the wire, l the length of the wire, μ the permeability of the wire metal, ρ its resisitivity, and ξ a factor (usually taken to be about 3) which specifies the destructive effects of intracoil interference. Note that the SNR goes like $B^{7/4}$, since $\omega_L = \gamma B$.

This suggests several optimizations that might be possible for NMR quantum computers, which ordinary NMR spectrometer manufacturers cannot utilize (Ger): for example, increased sample size is possible since we are not analyzing rare materials, nor are we working at such a high field that the homogeneous volume is constrained to be very small; this results in an increase in the cross-sectional area of the coil. For example, increasing the linear dimension of the sample by 10 would therefore result in a 100-fold increase in signal strength. Another possible optimization is that we can use soft pulses for very precise narrowband excitation, and address only a few spins with known chemical shifts – this suggests that we could make the Q of our coil extremely large,

another win in terms of signal strength. One notable area where we are working contrary to the majority of spectrometer designers is in the strength of the field: while M_x is proportional to B, the strength of the bias field, we are purposefully choosing to use lower fields for ease of implementation (at low frequencies, we can take advantage of software-radio style designs), ease of fabrication (we can use small permanent magnets, as opposed to large superconducting magnets), and ease of operation (the NMR spectrometer can fit on a tabletop, rather than requiring an entire gymnasium). In terms of signal-to-noise ratios, the effort required to raise the field by a small factor seems better spent on designing other parts of the spectrometer. As it turns out, at the current moment the dominant problem in our homebuilt NMR spectrometer is the large inhomogeneity of our biasing magnetic field; this is discussed in **VI.1.ii**., as well as one possible solution to the problem, as implemented by the author.

Not all of the signal-to-noise improvements in quantum computing have to do with the mode of detection. One can also try to prepare the state in different ways, such that less of the already-rare thermal polarization is lost through pure-state averaging.

One can take advantage of the fact that the gyromagnetic ratio of the electron is three orders of magnitude larger than that of any nucleus: $\gamma_e/\gamma_n \sim 1000$. It is possible to perform a pulse sequence called INEPT, which transfers the relatively vast polarization of the electron to the nucleus, but this takes special microwave hardware, since the electrons have a Larmor frequency that is correspondingly three orders of magnitude greater than that of the nucleus. Putting INEPT stages in both the preparation and readout stages of a quantum computation results in a 10^6-fold increase in signal strength (Chu98a).

Another interesting idea which might be feasible is the use of optical pumping to put electrons into highly polarized states, followed by the transfer of the resultant polarization to the nuclei via fine-structure coupling. The canonical optical pumping experiment requires a three-state manifold {A, B, C}, where A and C are low-energy states and B is a high energy state, and transitions between A and C are forbidden due to selection rules. If a particle in state A is irradiated at the resonant frequency ω_{AB} for the $A \leftrightarrow B$ transition, then it will undergo a transition to state B, then back to A, and so on. Occasionally, however, the excited state B will spontaneously decay into the low-energy state C, and the system will remain in that state, since there is no stimulated excitation to pump it back up to state B, and since the direct transition to state A is forbidden. Thus any ensemble of such 3-state systems, if suitably irradiated for a sufficient length of time, will eventually end up in the state C!

Although it has never been done before, it might be possible to perform spin-specific optical pumping on computationally interesting molecules, in which case it would be easy to extend an NMR quantum computer to many qubits, since the exponential decrease in signal due to the thermal constraints on qubit number could be obliterated by performing optical prepolarization. A simple optical pumping experiment has been demonstrated by Pines, who dissolved hyperpolarized ^{129}Xe (which in turn had been polarized through spin-spin interactions with optically pumped rubidium vapor) in water, causing hydrogen nuclei to become polarized due to the nuclear Overhauser effect (Pin96); polarizations of O(1/2) were achieved in this fashion. (Interestingly, this idea is now being prototyped for use as an MRI contrast agent in humans.) A macroscopic version of this idea, which is perhaps more amusing than practical, is to use a Stern-

Gerlach apparatus to iteratively separate polarized spins, perhaps by using mechanical means to partition differently polarized parts of the flowing fluid (Ger).

"Beware of bugs in the above code; I have only proved it correct, not tried it."
– Donald Knuth

V. The Art of Quantum Computer Programming: System Design

V.1. A Quantum Computer Language

V.1.i. High-level language: a case study

A few defintions are in order. A *register* is a collection of spins, which we can treat as a binary number – or, in general, a superposition of binary numbers. In order to shorten programs, we may occasionally use pulse sequences which compute correctly, but scatter the answer deterministically about the quantum system: for example, after performing one particular version of the Fourier transform, we might find that the digits of the answer, expressed as a binary number (or superposition thereof), are in reversed order (compared to the original labeling of qubits within the register). Rather than waste precious pulses swapping the qubits into the correct order, we can just relabel the spins, and continue the computation under the new labeling. It may be helpful to refer to a collection of qubits as a *semantic register*, which is independent of the actual spins upon which the quantum information happens to reside at any particular point in the computation. We refer to the list of spins on which a semantic register resides as the *chemical indices*. Throughout a computation, the semantic register retains its integrity, but the chemical indices might change as lazy computations operate upon the qubits, moving the information in a semantic register from particle to particle. This abstraction layer is important for understanding the model compiler code given in **Appendix A**.

Several people have constructed simple sets of commands that suffice to do certain computations on particular quantum hardware platforms. One relatively complete example is Preskill et. al.'s instantiation of Shor's factoring algorithm, implemented on a linear ion trap (Pre96). This team developed an algorithm that allowed the factoring of a K-bit number in $O(K^3)$ time, using $5K+1$ qubits. They focused on the problem of creating a simple implementation of modular exponentiation on a quantum computer, since that is the bottleneck of Shor's algorithm; their method uses about $72K^3$ quantum gates (equalling $396K^3$ laser pulses) to implement. For a 130-digit number (i.e., the best that current classical computers can handle in early 1999) this amounts to over 10^{10} laser pulses.

This team designed pulse sequences that implement a full adder of a quantum number to a classical number, with several variations involving various control qubits upon which the adder can condition. Using these adders, they implemented multiplication of quantum numbers by classical numbers, also utilizing operators that condition on control qubits. With these modules in hand, they constructed a pulse sequence for modular exponentiation of a classical number to a power specified by a quantum number, using repeated squaring. The effective construction is an assembly-language version of Shor's factoring algorithm. Matlab code which compiles several of these operations down to elementary operations is given in **Appendix A**; with slight modification and the addition of a simple parser, the simulation code in **Appendix B** could be used to construct a full quantum computer simulator. Some people have created very complex C/C++ based compilers (Tuc98) and simulation environments (Opn), and it is likely that due to

the relative simplicity of simulation compared to implementation, this will be a popular avenue of exploration for many.

V.1.ii. System architecture and low-level control: QOS

To understand the system architecture, and therefore the low-level machine language of the quantum computer, it is important to have a working knowledge of the hardware of the Mark I quantum computer, as described in **VI.1.** We will mention only the barest overview of the hardware in this section, because we want to motivate the hardware design choices properly in part **VI.** Therefore one reasonable partition is to consider the digital infrastructure and system architecture now, since that is essential to understanding how QOS (the *quantum operating system*) works, and to go over the analog issues later, in part **VI.**

The core of the hardware comprises two Hewlett-Packard modules, the HPE1437 (a 20-MHz, 23-bit (raw data, 32-bit if processed), 110 dB SFDR A/D converter with 8 MB RAM, equipped with software-selectable center frequencies, filtering capabilities, data types, triggering modes, and data acquisition strategies) and the HPE1445 (a 40-MHz, 13-bit, arbitrary waveform generator with 3.25 MB RAM, equipped with software-programmable waveform choreography, triggering modes, and filtering options) (HP). The HPE1445 module generates a 40 MHz clock that is divided down to provide clocks for the HPE1437 and for two continuous-wave (CW) generators that are used to mix the software radio components up to the Larmor frequencies for spins at 1 T; this clock synchronization is essential for providing phase coherence throughout each NMR experiment. Both HP modules rest within a VXI mainframe that has an ISA interface to a PC, which we use for all high-level control, compilation, and signal processing.

The basic signal path is as follows: the output waveform from the HPE1445 module is split into two parts, one for each of the two species of interest, ^{1}H and ^{13}C. Each channel is then mixed up to that species' particular Larmor frequency, using each of the two CW generators. The two channels are then mixed together before being amplified and sent to a coil surrounding a sample in an electromagnet. After the transmit signal terminates, a pin diode switch changes state, so that the coil is connected to the receive signal chain, instead of the transmit signal chain: if the transmit and receive chains were ever simultaneously connected, the sensitive front end of the receive chain would be instantly destroyed by the tens of watts of power emerging from the transmitter. The return signal is amplified, then split into separate channels for downmixing, then filtered and recombined for digitization by the HPE1437. The output level of each NMR species' transmit channel, as well as the state of the pin diode switch, is digitally controlled by a Hitachi SH-1 microprocessor board. Details of the Mark I quantum computer analog signal chain implementation are given in (Mag), the Master's thesis of the designer of the analog signal chain, as well as briefly described in **VI.1.i.**

Programs written in the *q programming language*, designed by the author to be Matlab-compatible (in the event that time allowed for the construction of a simulator of the system), are parsed by QOS (running on a PC) into waveforms that are sent to the HPE1445, at the beginning of each experiment. QOS also converts the *q* program into a series of timed digital transitions for execution by the Hitachi microcontroller, so that changes in the transmit switch and pin diode switch states are synchronous with the waveform transitions in the HPE1445. QOS also sets the

values for the currents flowing through any available shim coils in the system, as well as a plethora of other parameters which determine the system's behavior. In this section we will briefly discuss the instruction set and the program architecture; the *q* instruction set is elaborated upon in great detail in **Appendix D**, and QOS itself is examined in **Appendix E**.

An artificial *q* program designed to use every aspect of the *q* programming language is explored in the following paragraph, with detailed comments explaining what each instruction means. (The *q* files that perform Grover's algorithm, with temporal averaging and state tomography, are given in **Appendix D**, to clarify the constructs of the *q* language in a practical example.) QOS performs a round of parsing on the *q* program, then sends all system parameters to the appropriate modules (shim boards, HPE1437, HPE1445, Hitachi microcontroller). QOS then runs the parsed *q* pulse sequence through three rounds of elaboration – one to calculate the start, end, and duration of each pulse, a second one to actually synthesize voltage values and load them into HPE1445 segments, which are then arranged into waveform sequences, and a third to create the switch/pin diode transitions for the Hitachi microcontroller. Perhaps at this point it is best to go through a sample *q* program:

Comments.

```
%
% This denotes a comment, just like in MATLAB.
% Artificial q program, does nothing really.
% (C) Copyright blah, blah, blah. Who cares if you steal the code – you don't have a quantum
% computer to run it on anyway. Like it's going to do you any good.
%
```

Beginning of the program.

```
Qbegin()
```

Name of the program, sample, and version – for convenience of the human quantum computer operator.

```
Qname('g2|11>cd')
Qsample('13CHCl3')
Qversion('1.0 esb')
```

Decoupling style to use, during J-coupling and waiting periods.

```
Qdecouplingstyle('minimal')
```

General hardware and software modes of operation.

```
Qmode(verbose, nosimul, notcontinuous, triggered, armed, neitherinstantreplay,
specifyduration,specifyJtimes)
```

Which hardware modules to use and which to disable, for the current experiment.

```
Qmodules(no1437, 1445, hitachi,noshims)
```

Attenuation settings for the two channels.

```
Qrfsettings(2048,0)
```

Timing parameters for microcontroller synchronization.

```
Qhack(2e-6, 1e-6, 5e-6)
```

Digitization parameters for specifying HPE1437 behavior.

```
Qdigitize( 0.37, 5, 9, 2048, nodecimate)
Qdatatype(complex,32,fft)
```

Shimming parameters.

```
Qshims(4,0.01,0.005,0.003,-0.01)
```

Number of spin species, and relevant spin properties.

```
Qnumqbs(2)
Qspecies(1,'1H', 26.7510e7, 42647100.0, 46299000, 1, 0.00002)
Qspecies(2,'13C', 6.7263e7, 10723000.0, 12700000, 0.0159, 0.0005)
QJcouplings([0 215; 215 0])
```

Pulse sequences.

```
Qx(1,147,square)
Qy(2,140,square)
Qwait(500e-6, dec, 0)
Qx(1, 90, gaussian, .25, .3, 0.00005)
Qx(1, 90, halfgaussian, .25, .3, 0.00005)
Qx(1, 40, arbfreq, 0, 100000, 1)
Qp(1, 50, 0, 45000000, 1, square)
Qp(1, 50, 0, 41299000, 1, gaussian, .25, .3, 0.00005)
Qp(1, 50, 0, 41299000, 1, halfgaussian, .25, .3, 0.00005)
Qp(1, 50, 0, 41299000, 1, arbfreq, 0, 42299000, 1)
Qsynch(leftjustified)
 Qx(2, 10, halfgaussian, .25, .3, 0.00005)
 Qx(1, 10, halfgaussian, .25, .3, 0.00005)
Qendsynch()
Qsynch(rightjustified)
 Qx(2, 10, halfgaussian, .25, .3, 0.00005)
 Qx(1, 10, halfgaussian, .25, .3, 0.00005)
Qendsynch()
Qsynch(centered)
 Qx(1, 100, halfgaussian, .25, .3, 0.00005)
 Qx(2, 10, halfgaussian, .25, .3, 0.00005)
Qendsynch()
Qp(1, 50, 0, 41299000, 1, gaussian, .25, .3, 0.00005)
```

Data acquisition command.

```
Qread(100e-6,dec,0)
```

End of program.

```
Qend()
```

We will say no more on QOS at this point. Again, for the interested reader the actual commands of the *q* programming language are described in some detail in **Appendix D**, and some of the routines that make QOS operate are detailed in **Appendix E**.

V.1.iii. Control structures for QOS: an implementation, ROS

The following appealing scenarios have a common method of solution:

- A child inserts her hand into a little coil which sends harmless pulses of radiofrequency energy through her finger. It noninvasively extracts her blood oxygen content, glucose levels, and perhaps even scans her tissues for particular antigens, contaminants, or signal proteins (Fan) (Gru) (Nic). She leaves the doctor's office shortly afterwards, having only spent a few minutes before being given a clean bill of health.

- A food sample, inserted into a little coil, is scanned using multidimensional NMR techniques combined with nonlinear search methods. Grover's search algorithm on four qubits is compiled and optimized with respect to the various coupling and relaxation parameters, and in no time at all, we are performing a quantum computation on a piece of broccoli (Kno).

An eventual goal might be to embed QOS in a machine learning environment suitable for extracting the Hamiltonian of a spin system automatically, with very little prior knowledge. For example, one might begin such an experiment by scanning through the known Larmor frequencies of various spins, detecting what species were present. One might then run a multidimensional NMR experiment like COSY, extracting the various J-couplings between spins, as well as any other interactions that might be apparent. This is a very difficult task, requiring excellent hardware and software, since most real molecules in natural environments are difficult to differentiate from their surroundings. Much of the field of state-of-the-art NMR spectroscopy lies in interpreting the myriad peaks from a multidimensional experiment on extremely pure samples, as well as designing better pulse sequences (e.g., quantum computations) to extract detailed information about the structure of the molecular spin system. Performing useful NMR on natural samples is difficult to do, but the promise of noninvasive fast diagnostic and characterization techniques is immensely interesting to people in many disciplines.

More modest goals lend themselves to being automated. For example, shimming, while difficult to solve analytically, can be computationally approached with nonlinear search techniques. Finding the exact Larmor frequency of a given spin, or the exact duration of a $\pi/2$ pulse that maximizes the amplitude of the resultant transverse magnetization, can similarly be accomplished through automatic means. The envelope for a soft pulse, or the ideal structure of an iterated self-compensating sequence of pulses, can be designed simply by experimentation and nonlinear search.

I wrote a program, ROS, which does these kinds of optimizations. (It's called ROS simply because it was written one program later than QOS. I was hoping never to have to write SOS…) ROS can take a *q* program, and apply a certain *method* (Powell's direction set method, the downhill simplex method, brute-force search, conjugate gradient descent, cluster weighted modeling, etc.) so as to achieve a certain *goal* (maximum signal-to-noise, maximum T_2, maximum sharpness of a certain peak, etc.). Not all goals are compatible with all methods, of course. At the time of this writing, the available goals and methods were:

Goal: `browse`
Method: `easy`
Just browse through a range of parameters, saving the acquired NMR data for later use. The syntax for sweeping a variable through a particular range is best exemplified by a sample program. The following ROS script sweeps the angle α of an $X(\alpha)$ pulse from 1^o to 100^o in 0.5-degree steps, saving the data in separate files: For each value of α, a *q* program is synthesized and QOS is run on that *q* program. The value of α is then incremented, and the process is repeated.

```
Rgoal('browse')
Rmethod('easy')
% anything framed by Rprotobegin() and Rprotoend() is just copied into the q program
Rprotobegin()
  Qname('expt')
  Qsample('CHCl3')
  Qspecies(1,'1H', 26.7510e7, 42574931.880108, 35000000, 1, 0.001, 5)
Rprotoend()
% anything framed by Rsweepbegin() and Rsweepend() is swept through
Rsweepbegin( ALPHA, 1, 100, 0.5)
  Qx( 1, ALPHA, square)
```

```
Rsweepend()
```
Sweeps can be nested. This makes, for example, a COSY program completely trivial to implement in ROS.

Goal: `maxq`
Method: `simplex, powell, conjgrad`
The `maxq` goal indicates the desire to maximize the sharpness, or *Q* (using an analogy from the language of resonators in electrical engineering), of a certain peak; in the geometry of an NMR spectrum, this corresponds to the height of a peak divided by the FWHM (full width at half maximum). This roughly corresponds to maximizing the T_2 of a spin, and is a useful criterion for shimming. To compute the *Q* of a spin, given certain spectrometer parameters (shim currents, Larmor frequency, etc.), we do the following (code for this procedure is given in the **Quality (*Q* function)** section of **Appendix F**):

1) Synthesize a $\pi/2$ pulse program with specific values for the parameters being optimized.
2) Run QOS on the pulse program, and collect the data from the NMR spectrometer.
3) Perform a complex FFT on the collected data, and compute the power spectrum.
4) Linearly estimate the center frequency and the width of the peak of interest.
5) Perform Levenberg-Marquardt function-fitting, using a Lorentzian model, as is appropriate for the spin model discussed in **IV.1.i.**
6) Calculate the value of *Q* from the parameters of the Lorentzian, as the center frequency divided by the width parameter.

The Nelder-Mead downhill simplex method, Powell's direction set method, and the conjugate gradient methods of optimization are all explained in (Pre), as well as in the **Optimization Methods** section of **Appendix F**. The *Q* function to be maximized is very expensive (e.g., every function iteration requires a complete NMR experiment, plus lots of data analysis), and these three different strategies may offer different degrees of performance in different situations.

Also, the *Q* function, being a physical measurement followed by lots of data analysis, isn't a symbolically differentiable function – so for the conjugate gradient method (`conjgrad`) we use polynomial extrapolation from a series of experiments to compute a numerical derivative. As it turns out, this method is competitive with the other two methods listed in this section, due to the smooth search space often encountered when optimizing simple physical systems. The downhill simplex method and Powell's method largely ignore the local, differential information which characterizes a smooth search space.

A sample optimizing ROS script is shown below, for the purposes of shimming the main electromagnet. The goal is to tune four shim currents until the *Q* is maximized. In this version, the conjugate gradient algorithm is used.

```
Rgoal(maxq)
Rmethod(conjgrad)
Rtolerance(0.01)
Rverbose(on)

% relevant directory paths used by ROS
Rqospath('C:\users\nmr\qos2\qos2___Win32_Debug\qos2.exe')
Rdatapath('C:\users\nmr\qos2\data')
% the delay, in milliseconds, between experiments – important to let the transients die out
Rdelay(500)

Rprotobegin()
```

```
 Qname('shimtest')
 Qsample('glyceri')
 % modules, modes, digitize, etc. (omitted for clarity)
 Qnumqbs(2)
 Qspecies(1,'1H', 26.7510e7, 42647100.0, 49299000, 1, 0.00002)
 Qspecies(2,'13C', 6.7263e7, 10723000.0, 12700000, 0.0159, 0.0005)
Rprotoend()

Rrangebegin(SHIM1, -0.02, 0.02)
 Rrangebegin(SHIM2, -0.02, 0.02)
  Rrangebegin(SHIM3, -0.02, 0.02)
   Rrangebegin(SHIM4, -0.02, 0.02)
    Qshims(4, SHIM1, SHIM2, SHIM3, SHIM4)
   Rrangeend()
  Rrangeend()
 Rrangeend()
Rrangeend()

Rprotobegin()
 Qx(1,147,square)
 Qread(500e-6,dec,0)
Rprotoend()
```

Goal: `extract`
Method: `cwm, phascorr`
Cluster-weighted modeling (Ger98) might serve as a way to very precisely extract the frequencies and T_2's of the various peaks in a spectrum; however, simple Levenberg-Marquardt function fitting with a sum of multiple Lorentzians seems to work just fine, and is just as (if not more) realistic, since the signal actually is a sum of multiple Lorentzians. Therefore the `cwm` module was purposefully not completed. However, different parameter extraction and modeling problems, which require a proper probabilistic technique for appropriately interpreting the data, might arise in the future.

Phase correction is a useful procedure which removes nonidealities in the signal chain and in the synchronization of various modules. Due to delays and irregularities in an NMR system, signals at a frequency ω will often emerge with an added phase $\phi = a\omega + b$, which is undesirable. The net result is a distortion of the signal $A\sin(\omega t)$, which becomes $A\sin(\omega t + a\omega + b)$. Applying a linear fit (e.g., using the pseudoinverse method (Pre)) to the remnant phase can yield the coefficients a and b, and thus the artifactual phase can be deleted. Phase correction permits averaging the results from different experiments, as well as comparing phase-sensitive experimental results – e.g., telling apart the effects of X and Y pulses in two different experiments. The phase correction algorithm is described in more detail in the **Extraction Methods** section of **Appendix F**.

Here is a complete ROS script that uses the `phascorr` method to make a signal with initially unknown phase into a pure sinusoid ($\phi = 0$). Note that QOS is only called once in the execution of this script.

```
Rgoal(extract)
Rmethod(phascorr,real)
% if we want to use known values of a and b, we can specify them using the relative
% phase correction method:
%  Rmethod(phascorr,relat,0.393185,-5.68556e-007)
% All this does is subtract off the phase 0.393185 – 5.67556e-7 * w from the signal component
% at frequency w. This is useful for comparing different experiments, for example.
Rtolerance(0.01)
Rverbose(on)

Rprotobegin()
 Qname('shimtest')
```

```
 Qsample('glyceri')
 Qversion('2.0 esb')
 Qdecouplingstyle('minimal')
 Qmode(verbose, simul, notcontinuous, nottriggered, armed, notinstantreplay, specifyduration,
       specifyJtimes)
 Qmodules(1437, 1445, hitachi, noshims)
 Qrfsettings(2048,0)
 Qhack(2e-6, 1e-6, 5e-6)
 Qdigitize( 0.2125, 5, 9, 2048, nodecimate)
 Qdatatype(complex, 32, timedomain)
 Qnumqbs(2)
 Qspecies(1,'1H', 26.7510e7, 42647100.0, 49299000, 1, 0.00002)
 Qspecies(2,'13C', 6.7263e7, 10723000.0, 12700000, 0.0159, 0.0005)
 QJcouplings([0 215; 215 0])
 Qx(1,147,square)
 Qread(500e-6,dec,0)
Rprotoend()
```

V.2. Compiling for the NMR Quantum Computer

What we have described above is, in a sense, not compilation. It is the design of a straightforward and modular assembly language, with some support software to automate common experimental tasks. It is possible that as these paradigms mature, and quantum algorithm designers and hardware designers start to converge on useful primitives for understanding quantum computation, a high-level language which makes accessible the power of the hardware might evolve. It is important to remember that familiar languages like C and Java were not written right after the transistor was invented; a great many technological choices, architectural biases, and aesthetic decisions resulted in the programming languages we use today. It would be stupid to impose too much intellectual infrastructure on the frothing sea of free-ranging exploration which dominates all parts of the field of quantum computing at the present moment.

Nevertheless it's interesting to speculate on what practical quantum programming languages might look like. One interpretation, which is akin to that outlined in the previous section, is that quantum compilation is the process by which a known algorithm, like Grover's search on six qubits, or the simulation of a certain system of fermions, is broken down into the appropriate gates (and then pulse sequences) for operation on a particular quantum substrate. This interpretation is substantiated by **Appendix A** of this document, in which code is given which compiles various arithmetic logic units (multiplexed half- and full-adders, with and without enable bits), comparators, Fourier transform operators, and conditional gates, down to elementary operations. The idea is to take humanly-comprehensible operations, like adding, multiplying, searching, and comparing, and to break them down into elementary operations which are like machine language for a particular quantum system. For example, here is the Matlab code that outputs the elementary operations for a full adder, which adds the classical bit `cbit` to qubit `inputbit` and stores the result in qubit `outputbit` with the carry in qubit `carrybit` (see **Appendix A** for explanations of the subroutines called in the following code):

Matlab declaration of the program
```
function program = fa(cbit, inputbit, carrybit, outputbit)
```

The case of adding a classical 1 bit
```
if (cbit)
```
Create a program string
```
 program = cnot(carrybit, outputbit);
```

Concatenate the next term of the program to the program string

```
 program = str2mat(program, nnot(carrybit));
 program = str2mat(program, cknot(2, [inputbit carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
```

The case of adding a classical 0 bit

```
else
 program = cknot(2, [inputbit carrybit], outputbit);
 program = str2mat(program, cnot(inputbit, carrybit));
end;
```

Another interpretation is that compilation is the process by which an arbitrary unitary matrix is decomposed into elementary operations. One proposal for doing this is to use the *CS decomposition theorem*, which states that given a $2N \times 2N$ unitary matrix U, one can express U as

$$U = \begin{bmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{bmatrix} = \begin{bmatrix} L_0 & \\ & L_1 \end{bmatrix} \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix} \begin{bmatrix} R_0 & \\ & R_1 \end{bmatrix}$$

where $D_{00} = D_{11} = \text{diag}([C_1, C_2, \ldots, C_N])$, $D_{01} = -D_{10} = \text{diag}([S_1, S_2, \ldots, S_N])$, and $C_i^2 + S_i^2 = 1$ for each i. In particular, D_{ij} contains the singular values of U_{ij}. The algorithm proceeds as follows: in each iteration, apply CS decomposition to the L_i and R_i from the previous step, resulting in unitary matrices that operate on smaller-dimensional spaces, until only complex numbers and D-matrices are left. The problem now reduces to decomposing the remaining D matrices into elementary operations, which can be done using a series of painstaking theorems described in (Tuc98). A freely available C++ program, Qubiter, has been implemented which performs this kind of quantum compilation (Tuc98b).

There is a flaw in this model, however, which is that in order to perform this decomposition, or any decomposition like it, one must initially store the matrix U, which takes up space exponential in the number of qubits upon which it is designed to operate, in a classical computer's memory. This defeats the entire purpose of the quantum computer. A better model is that tools like the CS decomposition are primarily valuable for prototyping small unitary building-blocks, to aid in the intuition for and design of more complicated unitary operations.

V.3. Simulating

Simulating quantum computers gives researchers a taste of what power may lie ahead, allows people to evaluate proven algorithms and test new ones, and offers the possibility of probing the dynamics of quantum systems. For would-be quantum computer programmers, simulation offers new methods of developing algorithms, such as genetically evolving new quantum algorithms and optimizing compiled sequences of quantum gates, using search methods. Given the current state of quantum computers, simulators are probably essential for progress to be made on understanding the computational power of complicated quantum systems.

In **Appendix B** are Matlab listings for a large number of quantum computation primitives and some sample code which takes advantage of them. The primitives include single-bit rotations about the *x*, *y*, and *z* axes, two-qubit scalar J-coupling, the C^kNOT gate, and a function which takes any gate U_b and returns the conditional gate $CU_{a,b}$, which performs operation U on register *b* if the bits in register *a* are all 1. Also amongst the primitives is a Kronecker tensor function which takes any *n*-qubit gate $U_{a..n}$ and embeds it in an *N*-qubit gate $U'_{A..N}$, such that U' operates

on qubits *a*, . . . *n* just like U does, and operates on all the other qubits with the identity operator. This is useful if one is representing a quantum system by a large density matrix labeled by lexicographically ordered states (see **II.2.** for examples), and wishes to apply some operators (usually small, few-qubit operators) to certain subsystems within the large density matrix, or if one wants to check the decomposition of a large complicated operator into smaller, simpler ones. Several examples are given in this appendix, including verifications of NMR pulse representations for C^kNOT gates, Toffoli gates, and Hadamard transforms, and thorough simulations of Grover's algorithm and of temporal averaging. For clarity, we will go through Grover's algorithm (**III.1.iii.**) here, using the primitives declared in **Appendix B**. Note how `krqn`, the Kronecker tensor product, is used to embed single-qubit operations into operators on the complete two-qubit density matrix, for convenience in numerical evaluation.

Make a Walsh-Hadamard transform (up to a global phase)

```
H = krqn(2,1,Y(pi/2)*X(pi)) * krqn(2,2,Y(pi/2)*X(pi));
```

Let's say we want the function used in the Grover's search to return the fourth term – that is, C = diag([1 1 1 -1]) (up to a global phase)

```
C = krqn(2,1,Y(pi/2)*X(-pi/2)*Y(-pi/2)) * krqn(2,2,Y(pi/2)*X(-pi/2)*Y(-pi/2)) * ZZ(pi/2);
```

Inverting about the average uses the operator D = WPW where P = diag([1 -1 -1 -1]) (up to a global phase)

```
P = krqn(2,1,Y(pi/2)*X(pi/2)*Y(-pi/2)) * krqn(2,2, Y(pi/2)*X(pi/2)*Y(-pi/2)) * ZZ(pi/2);
```

The operators for the Grover's search:

```
D = W * P * W;
U = D * C;
```

Running the Grover's search on starting input [1 0 0 0]:

```
% Grover's algorithm, as implemented in (Chu98b):
U * W * [ 1 0 0 0]';
```

The output is [0 0 0 1], which is of course the correct answer. Many people have written similar product-operator packages for their personal use, or for sharing within the NMR community; one of the "most retyped pieces of NMR code in recent years" is the POMA package, which includes most of the product operator formalism in the Mathematica language. It can be downloaded at (Bmrl), and the original paper can be found in (Gun).

In addition to the quantum computer simulators being developed by the OpenQubit group (Opn), there is at least one paper in existence on Monte Carlo simulation of quantum computers, using techniques devised for stochastic simulation of many-body systems (Cer97).

Finally, we note that there exists a comprehensive platform for simulating many aspects of NMR experiments, including dephasing and dissipation models, solid-state NMR behavior, Redfield theory, magic angle spinning, and many other arcane or difficultly visualized parts of the discipline of nuclear magnetic resonance. While product operator methods are fine for visualizing the logic of gate and pulse sequences, other relevant problems like modeling error correction, studying the decay of information encoded in multiple-quantum coherences, understanding the best distribution of information across qubits of different T_2's and coupling strengths, comprehending soft pulses and multiplet manipulations, understanding electron interactions and liquid-motional effects, and gaining insight into subtle points of ensemble

computing, require more sophisticated models. The package, called GAMMA, is written as a set of C++ libraries, and most likely has a somewhat steep learning curve (as suggested by the author's brief exploration of its power), but is versatile enough to, say, serve as a set of primitives for a forward model used by a machine-learning program that performs adaptive quantum compilation (see **V.1.iii.**). See (Smi) and (Gam) for more information on this program.

"Actually making something work can be the intellectual equivalent
of going to war: becoming stronger, chauvinistic, with a requisite
sense of destiny and patriotism – without all that, it is too easy just to give up…"
– E

VI. Hardware Implementation of the NMR Quantum Computer

VI.1. Hardware Overview, Mark I

VI.1.i. Spectrometer

Spectrometers capable of quantum computation have traditionally (in their short tradition!) used samples like alanine, ^{13}C choloroform, and freon; standard NMR preparations typically use submolar samples dissolved in acetone or some other solvent, often with heavy water for frequency-locking purposes. Most existant spectometers are incredibly complicated, with hardware schematics that go on for hundreds of pages, with millions of individual parts that contribute to a incredible amalgamation of hardware for radiofrequency synthesis, broadband amplification, ultrastable power generation, and computational interpretation of the signals (Var).

A spectrometer capable of performing quantum computation must have precise frequency resolution, a magnetic field homogeneous enough for a T_2 sufficiently long to accomplish the computation at hand, and a sufficient diversity of spins such that control and readout is feasible. Precise care must be taken to shut out every possible source of noise, inhomogeneity, vibration, and fluctuation. An earlier attempt at a quantum computer, which is somewhat amusing given what we now know is required of a powerful spectrometer architecture, is given in (Cho95) (see also **Appendix Z**).

Let us begin with the selection of a chemical sample for the quantum computer. The data in the following table is taken from (BMRL96):

Species	^{1}H	^{13}C	^{19}F	^{31}P
spin	½	½	½	½
μ (in μ_N)	2.79284	0.70241	2.6887	1.13160
gyromagnetic ratio γ (rad T^{-1} s^{-1})	26.7510e7	6.7263e7	25.1665e7	10.8289e7
isotopic abundance (%)	99.99	1.10	100	100
sensitivity (rel. to ^{1}H)	1	0.016	0.83	0.0663

Most other spins have tiny sensitivities, or have very small J-couplings, or require special-isotope compositions. (The sensitivity is proportional to $\gamma^3 I(I + 1)$, which is proportional to the strength of the signal seen by a coil, as stated in **IV.3.ii**.) The desire to create long polymeric chains for easily addressed molecular architectures suggests that the backbone must be that of an organic compound. Finally, we require the sample to be a liquid: although solid NMR quantum computing is being considered in many different paradigms (Kan) (DiV) (Cor), it will be omitted in this discussion. In a practical sense, this restricts us to four main species: ^{1}H, ^{13}C, ^{19}F, and ^{31}P. Chloroform, $^{13}CHCl_3$, for example, has a J-coupling between H and ^{13}C of 215 Hz, a fairly large value as J-couplings go, and therefore the peaks are easy to tell apart; the interaction time

required for a J-coupling operation is also sufficiently short that a complete quantum computation can be accomplished within the coherence time of an NMR system. Fluorine compounds have many appealing properties – including large J-couplings and enormous sensitivities: it is likely that fluorine-and-carbon polymers will be the molecules of choice for some time to come. Hydrogen, while having the highest sensitivity of any atom, tends to have very weak J-couplings with other hydrogen atoms, due to the poor symmetry breaking afforded by different hydrogen locations. Finally, ^{31}P has a moderately large gyromagnetic ratio and very strong J-couplings – often greater than a kilohertz – with nearby ^{19}F atoms.

Some browsing through the figures in the reference (Ann) suggest the following insights: anything that restricts atomic motion or orientation, or varies the density of electrons near a nucleus, or constricts the conformational states (gauche, trans, etc.) of a molecule, will have an effect on the spectrum, of at least a few Hertz. Steric hindrance, electronegativity, and reduced shielding can contribute to higher chemical shifts. *trans* atoms tend to have much higher J-couplings than *cis* atoms, when considering molecules with carbon-carbon double bonds. Often there will be many J-couplings, some small and some large; all must of course be taken into account when considering how to allocate qubits for a quantum computation.

Much insight can be gained by studying the influences of ring configuration, the presence of nucleophiles, the concentration of relevant spins, and the symmetry and stereochemistry upon the J-couplings between different atoms in a molecule. In **Figure 3** below are some sample molecules, along with their J-coupling values (Ann).

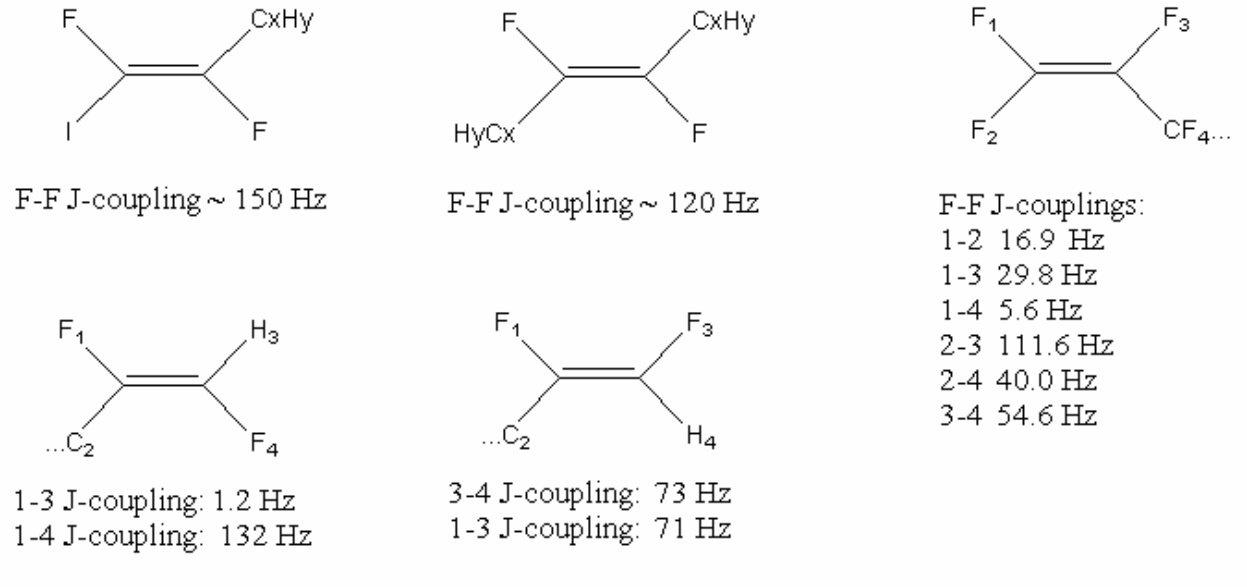


Figure 3: A few molecules and their J-couplings (Ann)

One can picture a carbon backbone with fluorine and hydrogen atoms located *trans* across each double bond, as a possible starting structure for making a quantum cellular automaton.

For the purposes of our simple experiments, we used carbon-13 chloroform. As it turns out, this molecule has a rather short relaxation time, due to the strong coupling of the Cl spins with the H

and ^{13}C spins, and so a better choice for a computationally interesting molecule might have been freon (or some other fluorine compound) – but time constraints prevailed, and only channels for ^{13}C and ^{1}H were constructed for the Mark I quantum computer (Mag).

The design of an NMR spectrometer begins by the choice of a method for detecting the spins. There are basically three strategies: detecting the radiofrequency energy emitted by a precessing spin (by using a coil, waveguide, or cavity), using a SQUID or force-feedback magnetometer to measure the actual amplitude of the magnetic field, and using optical interrogation methods to read out the nuclear state by observing the fine structure of the electronic spectrum. The first method is by far the easiest to implement and analyze, although the second has shown some promise: SQUIDs are by far the most sensitive magnetic-field detectors available, with resolutions of femtoTesla per $(Hz)^{1/2}$ and near-infinite bandwidth (which could, in principle, allow concurrent observation of electron-spin resonance (ESR) and NMR). A SQUID, or superconducting quantum interference device, comprises a superconducting interferometric ring with either one ("RF") or two ("DC") Josephson junctions along the circumference. By considering the quantum-mechanical interference of electronic supercurrents circling the loop, it can be seen that the output of a SQUID is a voltage $V(\Phi)$ which depends on the total magnetic flux Φ through the loop, and that V is periodic in Φ, with period equal to the magnetic flux quantum $\Phi_e \equiv \pi\hbar/e$. Since a SQUID can therefore measure DC magnetic fields, it can pick up the slowly-relaxing M_z component in an NMR experiment – something a coil cannot do. Typically, the noise in a SQUID is due to the Johnson noise of the ohmic current component flowing through the Josephson junctions, as well as the $1/f$ noises of defect trapping and flux vortex hopping; it has been estimated that SQUIDs offer SNR superior to that of a conventional NMR spectrometer at fields lower than 0.64 T (Gre98). The comprehensive review paper (Gre98) by Greenberg quantifies the reasoning behind these assertions, as well as some of the details of practical SQUID magnetometers. We avoid SQUIDs in the current design because they seem to violate the spirit of a tabletop, low-maintenance NMR spectrometer, and because there are enormous technical problems inherent in the implementation of a SQUID-based NMR spectrometer.

Even something as simple as a coil can be suprisingly subtle. For example, one can construct a coil setup which consists of two subcoils in series, one winding clockwise and the other winding counterclockwise, with the sample wholly contained in one of the subcoils, so that when a transmitter broadcasts energy to the spins, any residual excitation energy has less of an effect on the spins' output signal. Also, half the power emitted by a transmitting coil is completely wasted (since it's at twice the Larmor frequency, in the spin's rotating frame) – one can ameliorate this waste by using *slotted quadrature coils* which cleverly cancel this component of the radiofrequency radiation (Che89).

The analog signal chain, which was hurriedly described in section **V.1.ii.**, is depicted in **Figure 4**. The implementation of the mixing and amplification chain, as well as the PIN diode switches and the machined structural parts, is due to Yael Maguire (Mag). We will go into the hardware in some detail in this section. The Pentium PC, running QOS, synthesizes waveforms and sends the digital data representing the voltages to the HPE1445 in a very long ASCII string. Since the HPE1445 clock runs at 40 MHz, it is realistically only useful for synthesizing frequencies below 10 MHz (which puts us safely within the Nyquist limit for digital synthesis). Therefore QOS

synthesizes waveforms for ^{13}C and ^{1}H at the rather arbitrary frequencies of 2 MHz and 7 MHz, respectively, and the HPE1445 must broadcast the sum of the two signals, since it has only one output channel. As it turns out, having only one arbitrary waveform generator output channel causes most of the hardware headaches that we encountered throughout the design and construction of the Mark I quantum computer, and this motivates our use of a drastically different and simpler design for the **Mark II** quantum computer, whose design is given in the next section.

The HPE1445 output is first run through a DS-113 splitter-combiner (M/A-COM, 0.25 dB insertion loss, 400 kHz-400 MHz two way power divider), which divides the signal into two equal parts, one of which becomes the ^{13}C channel, and the other of which becomes the ^{1}H channel. The ^{13}C channel is run through a bandpass filter at 2 MHz, and the ^{1}H channel is run through a bandpass filter at 7 MHz, thus selecting the appropriate signals for each channel. The outputs of two continuous-wave generators (in our case a HP8647 running at 12.705 MHz, providing an intermediate frequency (IF) for the ^{13}C channel, and a HP83712 running at 49.576 MHz providing an IF for the ^{1}H channel) are mixed into the appropriate channels using two SRA-3MH mixers (Mini-Circuits, 4.77 dB conversion loss, 25 kHz-200 MHz, +9 dBm output). Each channel then is followed by a bandpass filter at the appropriate Larmor frequency – 10.7 MHz for ^{13}C and 42.6 MHz for ^{1}H – thus removing the useless mixing sidebands.

After passing through the bandpass filters, the signals are then ready for to be broadcast to the spins via the coils. Before entering the amplifier stage, each channel's signal passes through a digitally tunable RF2420 attenuator (RF Micro-Devices, DC-950 MHz, +17 dBm input), which can be set from 0 to 44 dB of attenuation according to the value of a 5-bit digital word. Each signal finally passes through an amplifier, a PIN diode switch, and into the coil surrounding the sample of interest.

The amplifiers include a 75-watt Amplifier Research RF amplifier, for the relatively insensitive ^{13}C spins, and a 4-watt Mini-Circuits amplifier, for the relatively strongly-responding ^{1}H spins. The PIN diode switch is necessary to switch the coil from the transmitter chain to the receiver chain: both the transmitter and the receiver need to be connected to the coil at certain times during the experiment, but if they were ever connected at the same time, the incredible power coming out of the transmit chain would destroy the sensitive receiver front-end. Most solid-state 1 ns GaAs absorptive switches can switch at most a few watts, but PIN diodes can switch hundreds of watts with similarly fast switching times. A PIN (*positive intrinsic negative*) diode has the special property that to microwave frequencies, it looks like a variable resistor. The resistance is controlled by a DC current flowing through the PIN diode: for example, when a biasing controller changes the DC current through the HPND-4165 PIN diode from 1 μA to 1 mA, the resistance of the PIN diode to RF energy changes from 2000 Ω to 10 Ω (HPRF). The PIN diode switch design is detailed in Hewlett-Packard RF and Microwave Application Note 1048, with additional details in AP922 and AP1049 (HPRF). The biasing currents for the PIN diodes are provided by Impellimax PIN diode drivers, which are described in (Imp).

The receive chain begins with its own set of PIN diode switches, connected to the coils for each channel, ^{13}C and ^{1}H. The front end amplifier for the receive chain is the UTC-101 (Avantek, 27.5 dB gain, 1.7 dB noise figure, 5-120 MHz, +16 dBm output), an extraordinarily low-noise (and

therefore high-expense) broadband amplifier. Each channel then enters a bandpass filter at the appropriate Larmor frequency, and then goes through two amplification stages and a downmixing stage (via the RF2612, a monolithic integrated IF amplifier/mixer). The signal then passes through a low-noise op-amp (the CLC425, by National Semiconductor), which frames the signal for the expected HPE1437 digitizing ranges. Finally, the two channels are combines using another DS-113 splitter-combiner, since we have only one analog-to-digital converter.

We used a Hitachi SH7032 microcontroller (as part of the SH-1 Low-Cost Evaluation Board, available from Hitachi) for multifarious digital control purposes: triggering the HPE1445 waveform generation process and the HPE1437 acquition process, adjusting the 5-bit digital attenuation settings, switching the continuous-wave generators from transmit-chain mixers to receive-chain mixers during the different parts of the experiment, and switching the coil from the transmit chain to the receive chain via the PIN diode switches. At the beginning of each QOS run, the PC would provide the Hitachi SH-1 with timing and state data for the various digital control elements, and then turn control over to the microcontroller, which would run the experiment until the data was ready to be read in to QOS at the end of the experiment. Code for salient parts of QOS is provided in **Appendix E**. A schematic of the Mark I quantum computer follows.

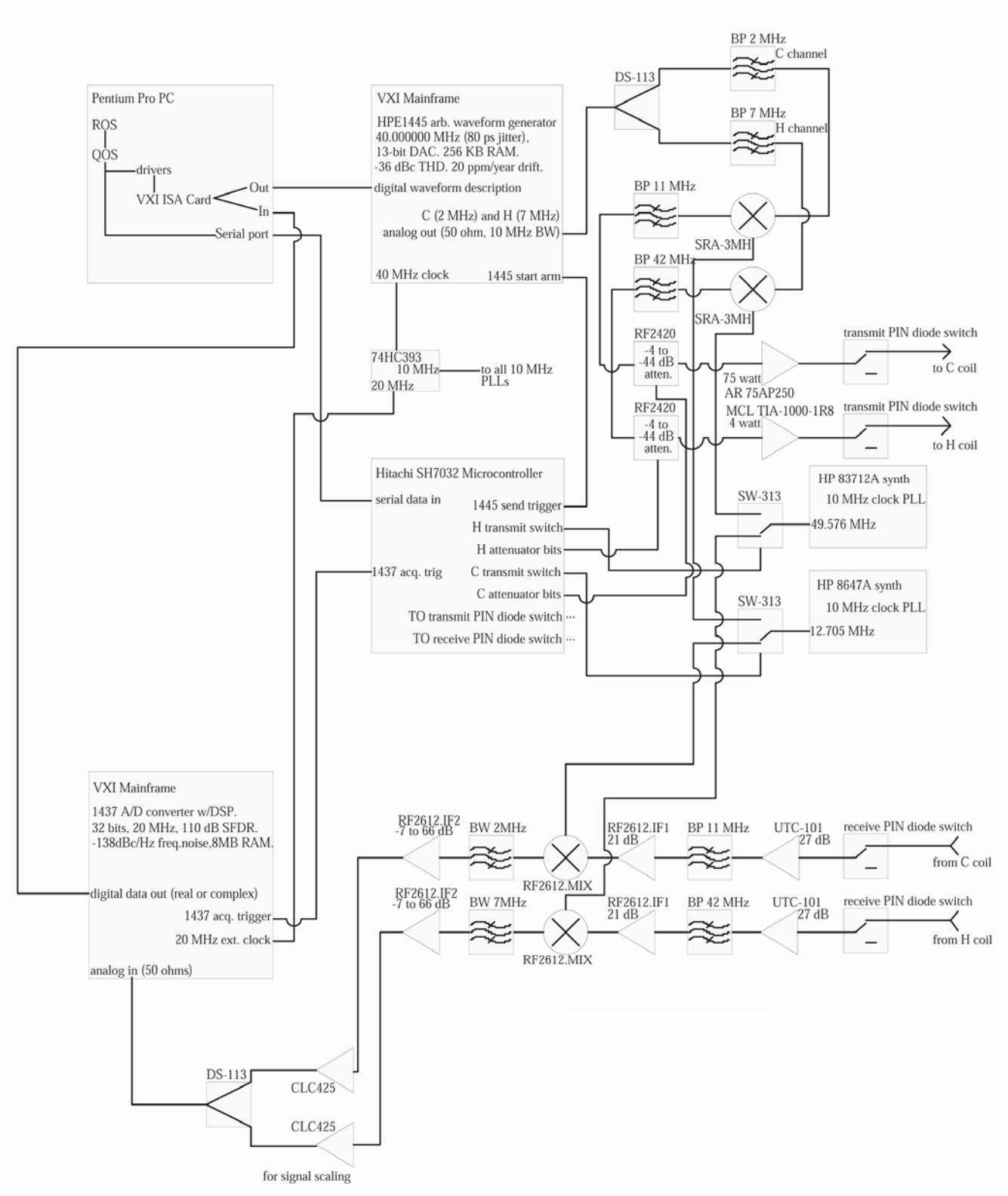


Figure 4: Mark I quantum computer block-schematic

Below is a photograph which gives a general overview of the quantum computer: visible is the electromagnet, the two CW generators, the controlling PC, and much of our test equipment. Not visible is the VXI Mainframe with the two HP modules, the custom-built RF hardware (Mag), the power electronics (including a cooling-water-interlocked 5 kilowatt power supply from Danfysik), and the chilled-water cooling pumps (from Bay Voltex and MagneTek).

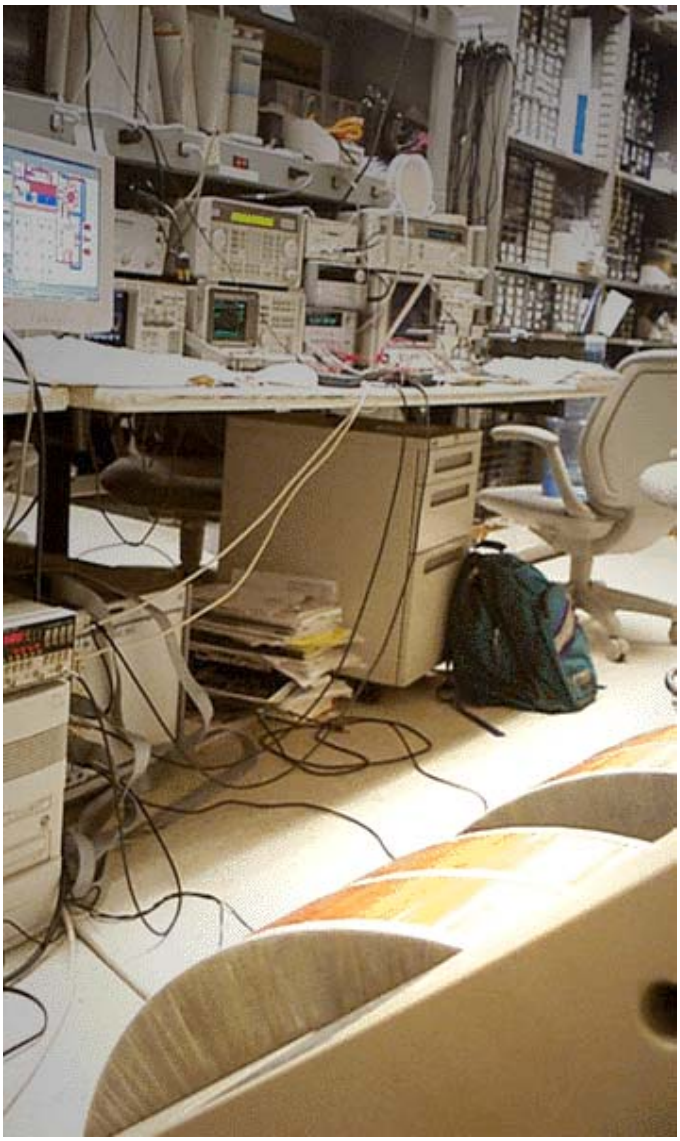


Figure 5: A photograph of the Mark I quantum computer

The magnet is a GMW associates custom-built electromagnet (model 3473-70), weighing 610 kg and dissipating 70 A across 60 V when running at a magnetic field strength of 2 T (and being cooled with 6 liters/minute of chilled 18^{o}C water). At a field strength of 0.5 T, it has a magnetic field homogeneity of roughly one part per 100,000 across a 3 mm sample tube. This is not good enough for quantum compuation, but allows some interesting NMR experiments to be performed.

Preliminary data suggests that the entire system functions very well: the signal-to-noise ratio of the ^{1}H (after applying a single $\pi/2$ pulse) is over 6 million, as can be seen from the following diagram (taken from a sample Visual Basic application that was modified to interact with QOS to assist with data acquisition and display). Some preliminary results suggest that the system can tell apart solutions of water with varying concentrations of glucose, and is sensitive enough to see the J-coupling in $^{13}CHCl_3$.

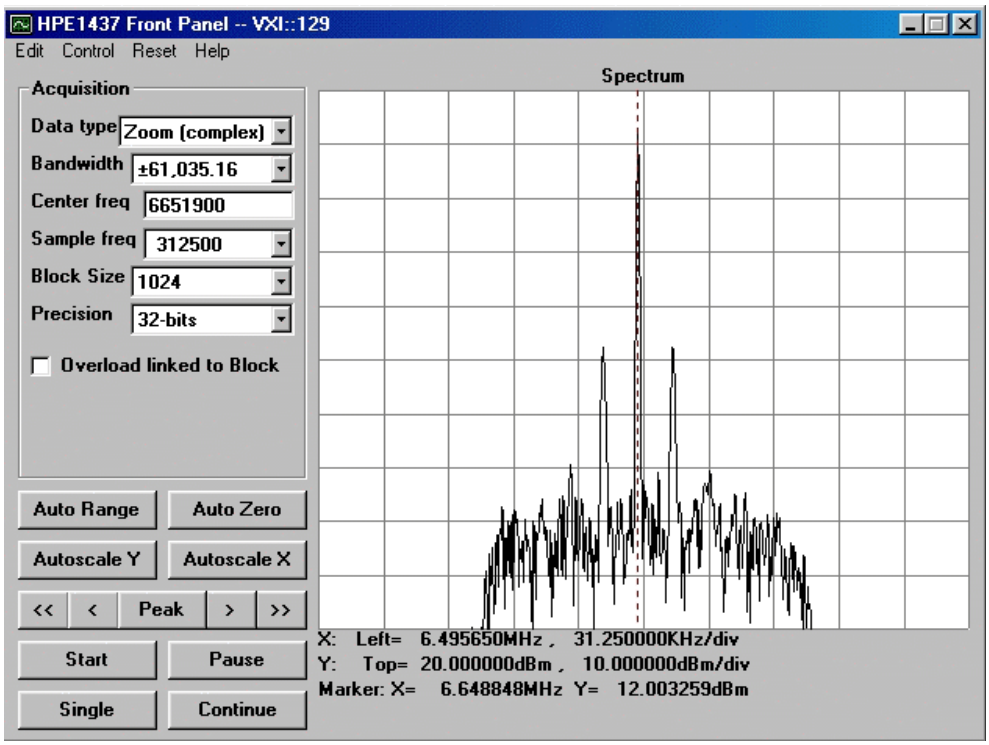


Figure 6: The signal-to-noise ratio for ^{1}H

One can perform state tomography to examine the state created by a certain sequence of pulses in an NMR experiment. By applying different pulse programs after completing the execution of a quantum computation, one can tip each element of the density matrix into an observable transverse magnetization signal, and then calculate the areas under the real and imaginary components of the peaks in the resultant frequency spectra. These amplitudes and phases can then be normalized in order to find the populations of the density matrix. With standard spectrometer technology, one can perform state tomography with ~5% error (Chu98a). We give code for performing state tomography in the context of Grover's algorithm in **Appendix B**. We have not yet tried to perform state tomography on our quantum computer, because the magnetic field homogeneity is not yet good enough for performing a full quantum computation. The next important step in improving the NMR spectrometer is the design of a shimming system. We discuss this very important issue in the next section.

VI.1.ii. Shimming

The 'ancient and honorable art of shimming,' as one expert practitioner calls it (Chm90), is essential for quantum computation due to the fact that nonidealities in the magnetic field can greatly reduce the lifetime of quantum coherences and blur features in the NMR spectrum. Shimming, or the modification of the magnetic field by precisely controlled electromagnets (or, in some cases, permanent magnets, mu-metal, or chunks of steel), can increase the homogeneity of magnets from one part in a million to one part in a billion, a 3-order-of-magnitude improvement over what precise machining and construction of a single magnet can provide.

The signal emitted by an NMR sample is proportional to the integral, over the sample volume, of the transverse magnetization density μ_x:

$$M_x = \iiint_V \mu_x(\omega_L(\vec{x}))d^3x$$

If $\omega_L(\vec{x})$ isn't uniform, then the observed signal will be a sum of the signals of many different *isochromats*, or collections of spins at the same frequency. A broadening of a waveform in the frequency-domain is associated with a shrinking of the length of the signal in the time-domain, since as the signal components at various frequencies dephase and become random with respect to one another, their sum tends towards zero. As discussed in (Lloy), this is a T_2 effect, caused by relative dephasing of an ensemble of spins, as the mutual information between the spins and the magnetic field increases, and the ensemble of spins acquires entropy. Since

$$\omega_L(\vec{x}) = \gamma \left|\vec{B}(\vec{x})\right|,$$

this means that maintaining a homogeneous magnetic field is of paramount importance. In particular, in order for a quantum computation to be possible, the dephasing time T_2 must be at least as long as the computation time – so in order to even see J-coupling, we must insure that T_2 is on the order of 1/J.

In a region without currents, a magnetic field obeys Laplace's equation, $\nabla^2\vec{B} \sim \nabla^2 B = 0$, which can be solved in terms of spherical harmonics. (We ignore the *x*- and *y*-components of the magnetic field, and consider only the *z*-component, which we refer to as *B*, in the following discussion.) Spherical harmonics are enumerated by indices (*l*, *m*), which are called the *order* and the *degree* respectively (just like the quantum numbers of the electron in a hydrogen atom). For $m = 0$, the *zonal* harmonics, one has cylindrical symmetry (i.e., the polar coordinate ϕ is irrelevant), which is appropriate for many magnets, where the dominant inhomogeneity is due to cylindrical end effects. The equation for *B* is then of the form

$$B(r,\theta) = \sum C_n (r/a)^n P_n(\cos\theta)$$

where P_n is the n^{th} Legendre polynomial, and C_n and *a* are parameters (Jack). (Setting θ to zero gives all the C_n as coefficients of an expansion in the variable *z*, since $z = r\cos\theta = r\cos(0) = r$. This is a useful fact, as we will shortly see.) In general, the complete solution for all (*l*, *m*) (including the *tesseral* terms, for which $m \neq 0$) is

$$B = \sum_{l=0}^{\infty}\sum_{m=0}^{l} C_{lm}(r/a)^l P_{lm}(\cos\theta)\cos(m(\phi - \psi_{lm}))$$

where P_{lm} is called the *associated Legendre polynomial*, and the last ϕ-dependent cosine term sinusoidally modulates the magnetic field as one traces a circle in the *xy*-plane (Chm90). All the

harmonics except for $l = m = 0$ contribute to the inhomogeneities, since a homogeneous field, by definition, has a magnitude which does not depend on any coordinates. Since the higher-order terms rapidly decrease in amplitude (for most solutions of Laplace's equation), adjusting a few terms should suffice to remove most inhomogeneities: this is the motivation for including between 10 and 30 shim coils with every modern NMR spectrometer. The value of the transverse angle ψ_{lm} complicates matters; this spherical harmonic phase can be interpreted as the orientation of the magnetic field due to the $(l, m)^{th}$ shim coil; in reality it is often specified using the ratio of currents flowing through two quadrature shim coils – that is, identical (l, m) coils, with one rotated 90^o about the z axis with respect to the other; the ratio of currents in the two coils then sets the transverse angle, just as a sinusoid with arbitary phase can be synthesized by adding together a proper combination of sines and cosines.

To make this picture concrete: one can setup a gradient with the (1, 0) shim, also known as the z-shim because in Cartesian coordinates the magnetic field is simply proportional to the z-axis coordinate (as discussed above). Since the magnetic field changes linearly with distance, the Larmor frequency changes linearly with z-coordinate as well; this allows simple imaging to be performed, since by specifying the currents flowing through the (1, 0), (1, 1), and (1, 1') shims, one specifies the Larmor frequency of the spins at a location given by z, x, and y respectively. Listening at a certain frequency is therefore equivalent to interrogating the spins in a certain area of space. This idea is the basis of MRI. (The (1, 1') shim is just like the (1, 1) shim but with a 90^o rotation in the xy plane; together these two coils provide a quadrature representation of the $l = 1$, $m = 1$ spherical harmonic component of the magnetic field.)

To experimentally detect which spherical harmonics of a magnet are present, one can probe it with a tiny NMR sample at different values of the coordinates, and watch for changes in the Larmor frequency of the spin. This produces a map of the magnetic field, but this technique is very difficult, due to the extreme mechanical sensitivity needed and the time required to scan the entire volume with sufficient precision. A simpler but more technical way is to image the magnetic fields by using the lineshapes of the spins of interest in the NMR spectrometer.

Spherical harmonics are orthogonal, so in theory one should be able to tune each coil once and be done with the business; in actuality they are not precisely orthogonal, and one is required to perform a Tower-of-Hanoi like algorithm, where one must adjust all lower-order-shims slightly whenever a particular shim current is changed. For a spectrometer with 10 shim coils, for example, one must conduct a search over a 10-dimensional space of current variables, trying to maximize the height and narrowness of the peaks in the spectrum. One can also try and minimize the presence of unexpected beats in an FID; a variety of extracted features can be used to make shimming automatic. Since there are nonidealities in the interactions between coils, this is a difficult problem, even for a computer, due to the number of possible local extrema in such a high-dimensional space. In fact, a small displacement of the sample from the center of the shim coils can introduce significant coupling terms between shim coils of different order, due to the fact that the NMR magnetization signal and the shim magnetic field are translated relative to one another, which somewhat obliterates the orthogonality of the spherical harmonics: a translation which seems trivial in Cartesian coordinates can result in very complicated changes in the spherical harmonics.

We summarize some human-appropriate strategies for shimming in (Shimnote). Shimming tesseral harmonics is difficult to accomplish without spinning the sample (Shimnote) or using field gradients, which pick out geometrically separated components of the sample (Chi). We don't want to spin our sample, and at this point we don't want to construct the hardware necessary to create gradients, so our design must be simpler and more powerful – and above all, more automated. Also, if we discard with the requirement that our shim coils be humanly tunable, and instead use a nonlinear search strategy to solve for the shim currents, then the nonlinearities inherent in nonorthogonal coils won't matter as much.

We propose a radical idea – to use planar coils, and to do the inversion completely in software, using nonlinear search techniques. Planar coils are extremely easy to fabricate, as opposed to the expensive machining and adjustment required to create spherical harmonic shim coils.

Schematics for some prototype planar coils are shown below. Their fabrication process is described in **Appendix G**, since planar coils are useful for many things (making tag readers for smart tabletops, making flat antennas or inductors for various purposes, etc.).

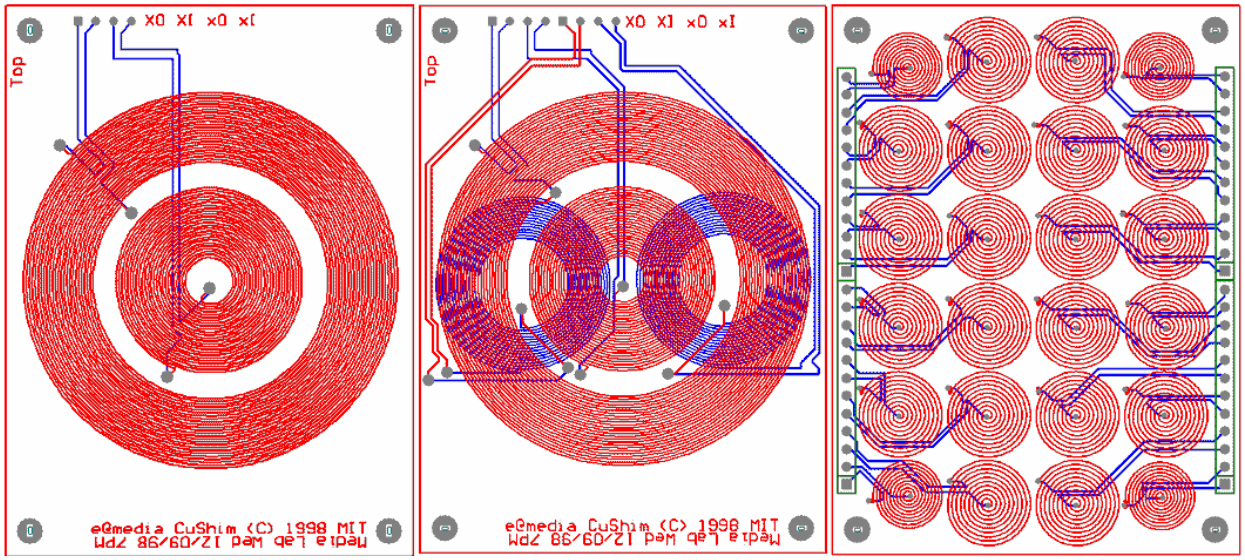


Figure 7: Prototype planar coils for shimming

The field created at the point $\vec{r} = (r, \theta, \phi)$ by a planar ring of radius a centered at the origin of the xy plane, carrying a current I, is described by the magnetic vector potential (Jack)

$$A_\phi(\vec{r}) = \frac{\mu_0}{4\pi} \frac{4Ia}{\sqrt{a^2 + r^2 + 2ar\sin(\theta)}} \frac{(2-k^2)K(k) - 2E(k)}{k^2}$$

$$\text{where } k = \sqrt{\frac{4ar\sin(\theta)}{a^2 + r^2 + 2ar\sin(\theta)}}$$

and where $K(k) = F(\pi/2, k)$ and $E(k) = E(\pi/2, k)$ are the elliptic integrals defined by

$$F(b,k) = \int_0^b \frac{1}{\sqrt{1 - k\sin^2\theta}} d\theta$$

$$E(b,k) = \int_0^b \sqrt{1 - k \sin^2 \theta}$$

Fast methods for numerically calculating these integrals exist (Pre), which suggests that it might be possible to perform a search on a forward planar-coil model to solve for the currents which correct for a certain inhomogeneous magnetic field. Given *N* planar coils of radius *a* centered at Cartesian coordinates $\vec{c}_i$ in space, each expressing the magnetic vector potential $\vec{A}(\vec{x} - \vec{c}_i)$ at the point $\vec{x} = (r \sin(\theta)\cos(\phi), r \sin(\theta)\sin(\phi), r \cos(\theta))$, we can derive the total field $\vec{H}_s$ due to all the shim coils, using

$$\vec{H}_s = \nabla \times \sum_i \vec{A}(\vec{x} - \vec{c}_i) .$$

where the components of the vector $\vec{A}$ are written in Cartesian coordinates for the purposes of computing this sum. The effect on the NMR signal, as expressed in the beginning of this section, is very complicated: let us denote the net magnetic field due to the shim coils as $\vec{H}_s$, and the bias field as the sum of the constant vector $\vec{H}_0$ and an error field $\vec{H}_e$. For a sample of spin-1/2 particles of density ρ in a sample volume *V*, and neglecting T_2/T_1 decay and other higher-order effects, the magnetization signal picked up by the coil is proportional to

$$s(t) = \int_V d^3x \rho \frac{\mu_0 \gamma^2 \hbar^2 \left| \vec{H}_s + \vec{H}_0 + \vec{H}_e \right|}{4kT} \cos(\gamma \left| \vec{H}_s + \vec{H}_0 + \vec{H}_e \right| t) .$$

This is a very complicated nonlinear functional of an already complicated nonlinear function, and a completely analytical method of solution would seem to be impossible. It is not clear how to find the error field, much less how to create a suitable distribution of shimming currents so as to cancel it.

Fortunately there is a way to solve the shimming problem with no further analytical work: just use nonlinear search methods. Given that an inhomogeneous field blurs the sharpness of a peak in the 1-D NMR spectrum, we can try to vary the shim currents in a given set of coils until the peak is as sharp as possible. This method is described briefly in section **V.1.iii.**, and the implementation is discussed at some length in **Appendix F**. At this point, we merely describe the shim hardware.

The goal of any active shimming hardware is to specify, with great accuracy, the currents flowing through a set of small electromagnets. Since our large electromagnet is capable of creating a 1 T magnetic field with part-per-100,000 homogeneity, we need to be able to create small magnetic fields on the order of 0.1 Gauss, and in order to get part-per-billion homogeneity, the error in these small fields must be on the order of 10^{-5} Gauss, roughly 10,000 times smaller than the Earth's magnetic field. We attempt to solve this demanding problem by using for each coil a 16-bit DAC, the AD420, capable of outputting between 0 and 20 mA of current, so that the least significant bit corresponds to 305 nA of current. A SPDT switch made of two electromechanical relays is used to select the polarity of the current flowing through each coil. For a 25-turn coil of diameter 5 cm, this creates a magnetic field with magnitude ~ 0.1 G (1 cm away from the coil center), so that the least significant bit corresponding to about 10^{-6} G. Thus the chosen hardware is (at least in theory) capable of creating the necessary fields. The actual

performance depends on many things, including the stability of the shimming system's power supply, the susceptibility of the electronics to external noise, and unavoidable noise imposed by the laws of thermodynamics.

The final shim board, designed to be digitally controlled from a single PIC16F84 microcontroller for ease of use, is shown in **Figure 8**, at roughly half-scale. The PIC controls all 32 relays using a serial-to-parallel shift register with latching outputs, the 74F673A (the narrow 24-pin DIP on the left side of the board, made by Fairchild Semiconductor). The PIC sends the 16-bit word describing the output current to all 16 AD420s, each of which reads in the word through a serial-to-parallel shift register – but only one AD420, selected via the MM74HC4514 4-to-16 line demultiplexer (the wide 24-pin DIP in the lower left corner of the board, also made by Fairchild), receives a rising-edge pulse which latches the word into the DAC. This permits full control of the entire board using a single PIC microcontroller, and vastly simplifies programming, layout, and assembly costs. The PIC code which performs the addressing and control is given in **Appendix G**.

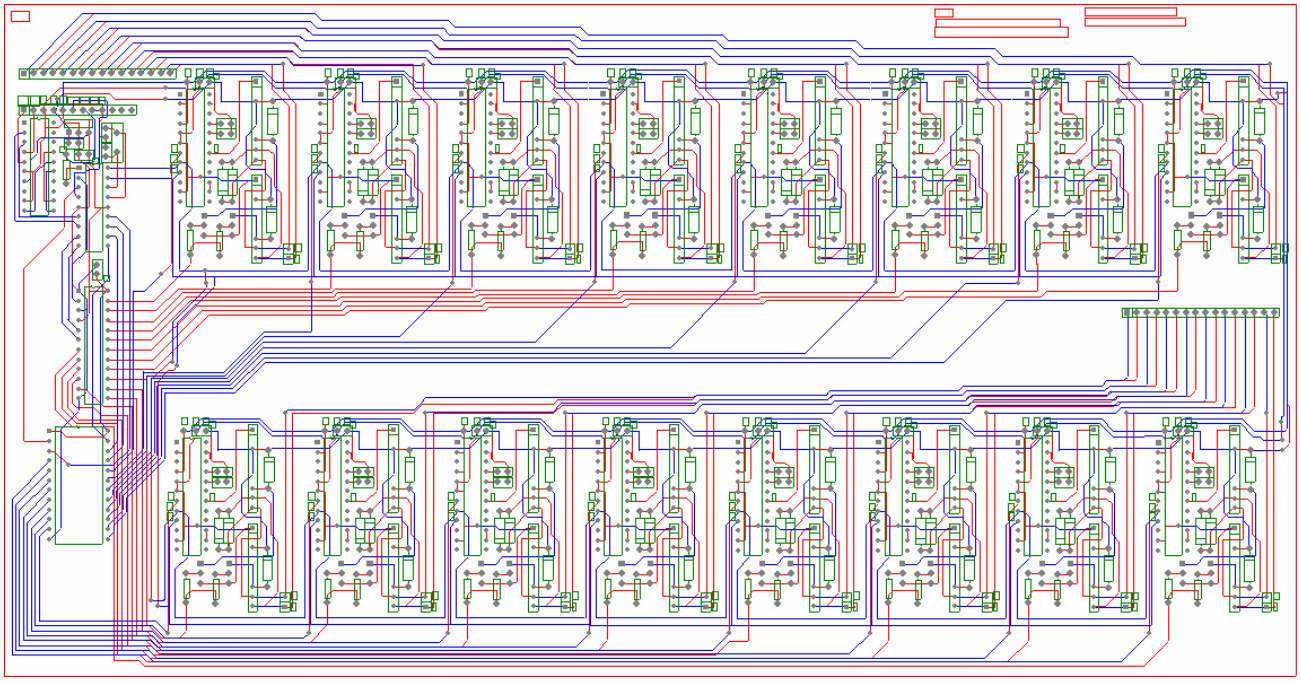


Figure 8: The shim board

At this stage in development, the shim board has been verified to operate, but no actual experiments have been completed. The software methods described in **V.1.iii.** have been shown to successfully shim orthogonal coils by (Mag) in collaboration with Issac Chuang at IBM Almaden; it remains to be seen how successful these computational methods are in the shimming of nonorthogonal coils.

VI.2. Hardware Overview, Mark II

VI.2.i. Testbed for the FPGA: the nanoTag reader

To minimize the necessary amount of control logic, I chose to implement the second-generation NMR spectrometer control system directly in digital gates. Using field-programmable gate arrays allows extremely precise allocation of digital resources down to the gate level, offers inherent parallelism and hundreds of pins of I/O for interaction with the outside world, lets the user precisely time synchronous events down to fractions of a nanosecond, is much cheaper than designing an ASIC (if needed, one can add a serial EEPROM containing configuration information to the FPGA, allowing it to bootstrap itself, or one can even take a wafer with a gate array imprinted on it, and simply perform a metallization step which indicates the appropriate connectivity), and allows many levels of description appropriate to the different parts of a complicated system.

An FPGA is a chip with combinatorial logic, lookup tables, wires, and flipflops in a 2-dimensional array. Each element has some control structure associated with it, telling that element how it is to be connected to the rest of the system, what its default states are, and so on. Some elements can be designated as local RAM, others can be treated as input pins, and still others can be combined into registers, accumulators, multiplexers, and other standard logical elements. When the FPGA is turned on, one must load a configuration string into it, in order to program the gates.

FPGAs can be programmed in many different languages, including the industry-standard ones for specifying digital circuitry, Verilog (Xil) (Arn) and VHDL (Arm). These hardware description languages allow the same logic (down to the register, wire, and gate level) to be represented independently of the underlying FPGA hardware (with its combinatorial logic blocks, ratsnests, longlines, pinwires, and other odd constructs). (AMS-HDL, which stands for analog and mixed signal hardware description language, will hopefully soon allow this power to be extended to analog circuitry as well (Sen).) I can write a DDS (direct digital synthesis) module in Verilog, and synthesize it for the Xilinx XC4010XL, PC84 package, speed grade –03, and then take the same Verilog code and implement it on a different brand of FPGA, using the synthesis tools for that chip. Many FPGAs also have schematic and finite-state-machine tools that allow logic to be designed in an intuitive way; for example, one might want to specify low-level pipelined mathematics hardware using register-transfer languages (RTL) and functional forms, while using the finite-state-machine tools for designing the computer's microcontroller.

A swept tag reader must display some of the same functionality as an NMR spectrometer. The idea behind a swept tag reader is that certain high-Q structures can absorb a significant amount of energy from a transmitter, and either thermally dissipate it or reradiate it isotropically. The transmitter (or equivalently, an adjacent receiver antenna) sees this effect as loading. If one sweeps through a range of frequencies, and notes what frequencies show significant loading, then one can identify the tags in the area (Tag). Tagged objects can thus be observed and/or tracked, and no power or computational structure on the tag itself is actually required – all of the sophisticated hardware is located on the tag reader.

I designed a tag reader using the XESS Xilinx Evaluation Board (Xes), which comprises a 950-logic cell, 10,000-gate Xilinx XC4010XL-PC84-03 FPGA, a 12-MHz clock, an Intel 8031 microcontroller, and some onboard RAM. Only the FPGA was used for this project; the microcontroller and RAM were disabled for simplicity. Eight pins of the FPGA were linked to the digital inputs of an 8-bit DAC, the AD7228, and 12 pins of the FPGA were linked to the digital outputs of a 12-bit ADC, the AD1671 (Ana). The unipolar DAC output was filtered with a 3-pole Butterworth filter with a cutoff frequency of 100 kHz, then converted to an amplified bipolar signal (using AD712/AD713 op-amps). This signal was amplified to 0.5 W using a crossover-distortion compensated push-pull amplifier (Hor) comprising two power transistors (the Motorola TIP31 and TIP32), and then routed into a hand-wound transmitter coil (40 μH) made out of copper wire. A receiver coil (converted from an old shim board!) was placed just under the transmitter coil, and the output of this coil was amplified to fit the $\pm$ 5V frame imposed by the input of the AD1671. Pictures of this assembly (no schematic or PCB layout was created) can be found below. The XESS board provided connectors linking the FPGA outputs to a VGA monitor and to a 7-segment LED display, as well as a parallel port connector for downloading the FPGA configuration bits from a PC.

Here is a picture of the nanoTag reader, seen from above. The red finned pieces are heat sinks for the power transistors, and the yellow rectangle is the FR4 epoxy supporting the receiver coil (which used to be a shim coil, see **VI.1.ii.**)

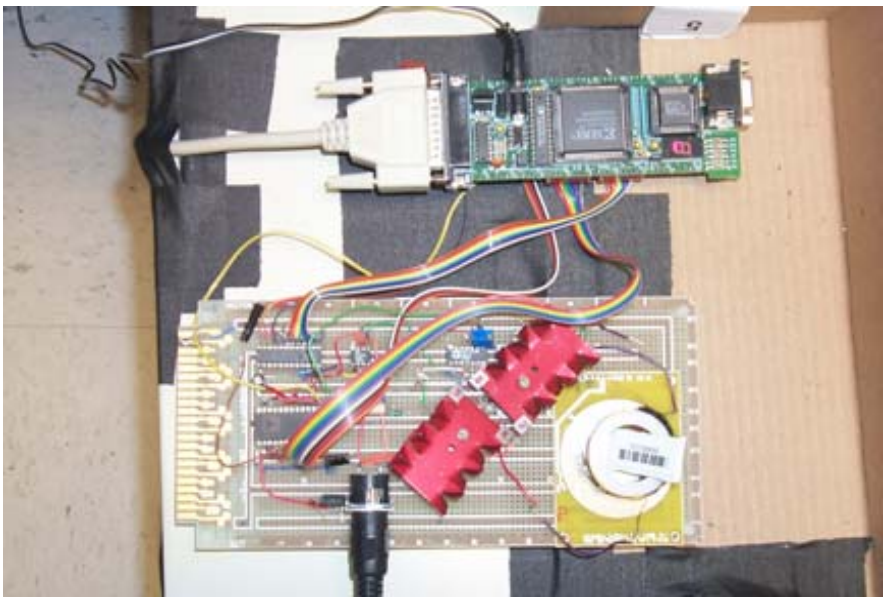


Figure 9: The nanoTag reader

I wrote Verilog code for a direct-digital synthesis (DDS) module, and implemented it for the Xilinx FPGA on the XESS board. (As it turns out, the Xilinx Core Generator tool, combined with the Xilinx Logiblox tool and the Xilinx Capture Schematic tool (Xil) reduce this code-writing effort to a few trivial mouse clicks.) A DDS contains four simple digital constructs: a clock, a frequency word, a phase accumulator, and an output amplitude (see the datasheet for the AD9850 (Ana) for more detail). At each clock tick, the phase accumulator is incremented by the

value of the frequency word, and the top few bits of the phase accumulator are then converted to a sinusoidal output amplitude via a lookup table. The result is a sinusoidal output with a precisely tunable output frequency, appropriate for performing a network analyzer task like sweeping through frequencies, or for performing a communication task like FSK modulation.

Verilog code is structurally like C, with functions, modules, and loops that are executed when certain registers (like the CLOCK) change (Xil). In Verilog code, a DDS might look something like the following (complete functional code for the nanoTag reader is given in **Appendix C**):

Given an external clock, divide it down to various multiples for convenience.

```
always begin
 @(posedge CLOCK)
 CLOCK_DIV65536 = 0;
 CLOCK_DIV64 = 0;
 CLOCK_DIV32 = 0;
 CLOCK_DIV16 = 0;
 CLOCK_INTERNAL_COUNTER = 0;
 forever begin
  @(posedge CLOCK)
  CLOCK_INTERNAL_COUNTER = CLOCK_INTERNAL_COUNTER + 1;
  CLOCK_DIV65536 = CLOCK_INTERNAL_COUNTER[15];
  CLOCK_DIV64 = CLOCK_INTERNAL_COUNTER[5];
  CLOCK_DIV32 = CLOCK_INTERNAL_COUNTER[4];
  CLOCK_DIV16 = CLOCK_INTERNAL_COUNTER[3];
 end
end
```

Latch the current output amplitude into an external DAC.

```
always begin
 forever begin
  @(posedge CLOCK_DIV32);
  DAC_LATCH = 0;
  @(posedge CLOCK_DIV32);
  DAC_LATCH = 1;
 end
end
```

If sweeping, increment the frequency word to increase the frequency.

```
always begin
 @(posedge CLOCK_DIV65536)
 freq_freq_word = 32'h70A07F;
 freq_word = 32'h15D867C;
 forever begin
  @(posedge CLOCK_DIV65536)
   freq_word = freq_word + freq_freq_word;
 end
end
```

Increment the phase by the frequency word, and look up the corresponding sinewave amplitude.

```
always begin
 @(posedge CLOCK_DIV64)
 phas_accum = 32'h0;
 DACword = 8'd127;
 forever begin
  @(posedge CLOCK_DIV64)
  phas_accum = phas_accum + freq_word;
  case (phas_accum[31:24])
    8'h0:DACword = 8'h7f;
    8'h1:DACword = 8'h82;
    8'h2:DACword = 8'h85;
    8'h3:DACword = 8'h88;
    8'h4:DACword = 8'h8b;
    .
    .
    .
    8'hfb:DACword = 8'h6f;
    8'hfc:DACword = 8'h72;
```

```
    8'hfd:DACword = 8'h75;
    8'hfe:DACword = 8'h78;
    8'hff:DACword = 8'h7b;
  endcase
 end
end
```

Note the key features: each `always` block runs in parallel with all the others, each `@(posedge CLOCK_X)` waits until that particular clock has a rising edge, and different blocks can run under different clocks to allow very complicated synchronous events to occur. This architecture is very simple, with no overarching finite state machine control structure or microcontroller.

The nanoTag reader exhibited some loading behavior when a 53 kHz magnetostrictive tag was placed inside the coil, but there was insufficient room on the FPGA for performing the frequency-space localization. Some simple ways to increase the amount of computational power include:

- placing the phase-to-amplitude lookup table in external DRAM, SRAM, or ROM/EEPROM
- storing only one quadrant of the sinewave, from $\theta = 0$ to $\pi/2$, and using a little bit of logic to compute amplitudes in the other three quadrants
- doing some of the computation in a general purpose microcontroller, DSP, or computer attached to the FPGA, relegating the FPGA mainly to the realms of fast digital control, interfacing, and reactive performance.

As we will see in the next section, all three of these ideas look very promising and very powerful.

VI.2.ii. A reconfigurable software radio portable NMR spectrometer: nanoNMR

We combine these digital design ideas with an analog design inspired by that of the Mark I spectrometer, resulting in the "nanoNMR" spectrometer.

Vi.2.ii.a. High-speed digital and analog design

High-speed digital and analog design are both mysterious fields, due to their great complexity. Analyzing spatially-distributed circuitry in indefinite surroundings is an intractable task, and the wisdom of the good high-speed engineer is hidden in cryptic rules-of-thumb, which require intuition and time to understand, but which usually make sense in hindsight. These rules-of-thumb connect the straightforward world of macro-models and transfer functions with the nonideal world of noisy power supplies, crosstalk between different lines on a board, $1/f$ noise from the environment, coupling between signals on different chips, temperature-dependent parameters, and incalculable but estimable parasitic resistances, capacitances, and inductances. For example, it usually makes sense to insert a few tunable elements in each stage of the design, for the practical reason that optimal behavior may not be exactly forecastable, given the tolerances of the parts used. Some of the rules which are worth keeping in mind are summarized in a worthy series of application notes by Analog Devices (AN342) (AN202).

A block-diagram schematic for the nanoNMR spectrometer is shown below. We briefly describe the four boards which comprise the nanoNMR system.

BrightStar Board (Bri)

This is a commercially available board which contains a Motorola PowerPC MPC823 chip, running a special version of Linux which fits into 2 MB of Intel BootBlock Flash RAM. The board comes with 16 MB of DRAM, has a 10BaseT port and multiple ports for serial, video, and USB data, and most importantly, contains a 16,000-gate Altera EPF6016 FPGA with 82 pins of I/O. Also on board is 128KB of SRAM which is mapped to certain pins of the FPGA.

Auxiliary board
The auxiliary board contains a 125 MHz oscillator which serves as the primary clock for the FPGA. The FPGA synthesizes digital representations of waveforms at the Larmor frequency of the spin of interest, as well as at the Larmor frequency minus 455 kHz (for the receiver board).

Transmitter board
The transmitter board receives digital data from the FPGA on the BrightStar board, synthesizes it into an analog waveform at the Larmor frequency using a 125 MHz 14-bit DAC (the AD9754), which then passes through a series of amplification and filtering stages. The final module is a CA2832C 1 watt power amplifier, which fits directly on the board; the output then passes through a PIN diode switch into a coil, just as in the Mark I quantum computer case.

Receiver board
The receiver board consists of the same low-noise front end as the Mark I quantum computer, but terminated with an AD9260 (16-bit, 2.5 MHz) ADC. We mix down using a digitally synthesized signal at the Larmor frequency minus 455 kHz, so that the final digitized signal is expected at a center frequency of 455 kHz. This avoids the $1/f$ noise associated with mixing down to baseband, while staying comfortably below the Nyquist limit for digital sampling.

Figure 10 shows the interrelations between these various modules, for a clearer picture of their functions.

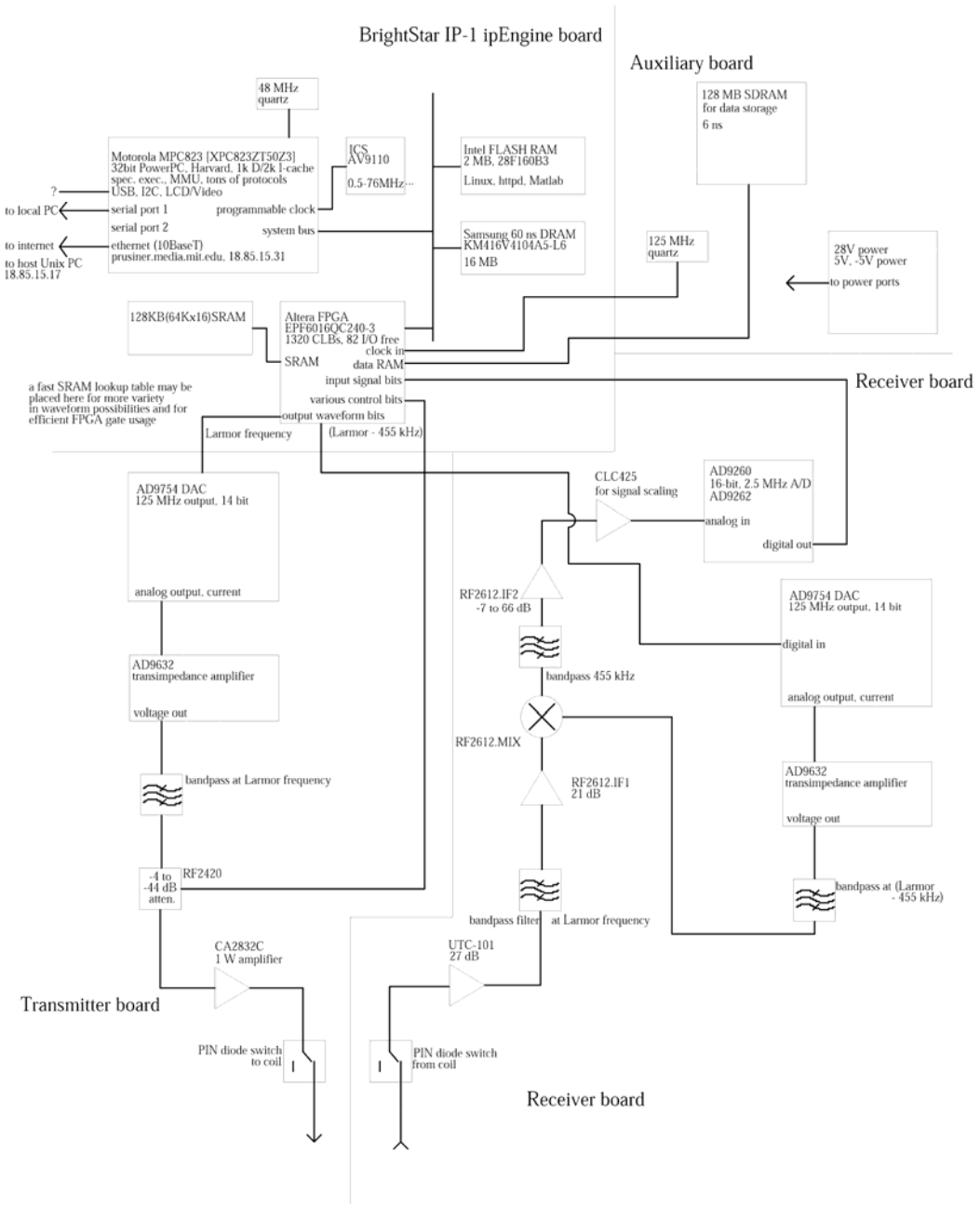


Figure 10: the nanoNMR block-schematic

As of May 7, 1999, only the transmitter board has been laid out in PCB form; the schematics and PCB photomasks for the transmitter board are included in **Appendix C**.

"There is a fine line between prognostication and lying…"
–E

VII. Conclusion

…and I would hesitate to do either, especially when writing something for an ostensible posterity.

Certainly the construction of a portable NMR spectrometer is a useful goal. And I would hazard to guess that quantum computation is bound to play an important role in many futuristic technologies.

I hope that this document has proven enlightening to the reader.

Appendix A: High-Level Compilation

This is Matlab code which compiles the various modules of Preskill code (Pre96), down to elementary logical gates. Since we stop the compilation at the CNOT stage, these modules are therefore appropriate for NMR quantum computing as well. An exception is the Fourier Transform module, which uses the CV and H gates, where CV is given in **II.2.**

C^kNOT:

```
function program = cknot(order, indices, outdice, maxorder, temp)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cknot.m
% apply Barenco's identity C[...mn],x = C[F,n],x C[...m],F C[F,n],x C[...m],F
% Implements controlledKnot
% Edward Boyden, e@media.mit.edu
% INPUT
%  indices     column vector of indices on which to operate
%  outdice     output index
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = length(indices); % contract L = order
if ( (nargin>3) & (maxorder<order) ) % the k is greater than allowed
   if ( (maxorder==1) & (order==2) )
      program = sprintf('CKNOT%s, %d', sprintf(' %d', indices), outdice);
   else % just do ordinary decomp
      program = cknot(L-1, indices(1:(L-1)), temp, maxorder, outdice);
      if( (maxorder==1) )
         prog2 = cknot(2, [temp indices(L)], outdice);
      else
         prog2 = cknot(2, [temp indices(L)], outdice);
      end
      program = str2mat(program, prog2, program, prog2);
   end
else     % maxorder>= order
   if (1==L)         % reached a CNOT
      program = cnot(indices(1), outdice);
   else
      program = sprintf('CKNOT%s, %d', sprintf(' %d', indices), outdice);
   end;
end;
```

CNOT:

```
function program = cnot(i,j)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cknot.m
% Implements controlledKnot
% Edward Boyden, e@media.mit.edu
% INPUT
%  indices     column vector of indices on which to operate
%  outdice     output index
% We do not compile the CNOT further because that's a job for the lower-level stuff.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%recall  C i; j = Uj^(-1) Vij Uj
%program = sprintf('Uinv %d', j);
%program = str2mat(program, sprintf('V %d %d', i, j));
%program = str2mat(program, sprintf('U %d', j));
program = sprintf('CNOT %d, %d', i, j);
```

Not:

```
function program = not(i)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% not.m
% Edward Boyden, e@media.mit.edu
program = sprintf('NOT %d', i);
```

Fourier Transform:

```
function program = ft(indices)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% ft.m
% Implements fourier transform
% Edward Boyden, e@media.mit.edu
% INPUT
%  indices     column vector of indices on which to operate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = length(indices);
program = sprintf('# Fourier transform on qubits %s', sprintf('%d ', indices) );
for i = L:-1:1,
 for j = L:-1:(i+1),
    program = str2mat(program, sprintf('CV %d %d %e', indices(i), indices(j), pi/2^(j-i)));
 end;
 program = str2mat(program, sprintf('H %d', indices(i)));
end;
```

Full Adder:

```
function program = fa(cbit, inputbit, carrybit, outputbit)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Edward Boyden, e@media.mit.edu
% add cbit to the value in the semantic inputindex to get the semantic outputindex
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (cbit)      % adding a 1 bit
 program = cnot(carrybit, outputbit);
 program = str2mat(program, nnot(carrybit));
 program = str2mat(program, cknot(2, [inputbit carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
else           % adding a 0 bit
 program = cknot(2, [inputbit carrybit], outputbit);
 program = str2mat(program, cnot(inputbit, carrybit));
end;
```

Multiplexed Full Adder:

```
function program = muxfa(cbitif0, cbitif1,selindex, inputbit, carrybit, outputbit)
% Edward Boyden, e@media.mit.edu
% if selindex is 1, add cbitif1, else add cbitif0, to inputbit with carrybit to get outputbit.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ( (cbitif0 == 0) & (cbitif1==0))  % both cases are 0, so selindex doesn't matter
 program = fa(0, inputbit, carrybit, outputbit);
elseif ( (cbitif0 == 1) & (cbitif1==1))  %"
 program = fa(1, inputbit, carrybit, outputbit);
elseif ( (cbitif0 == 0) & (cbitif1==1))   % --> add 0 or 1
 program = cknot(2, [selindex carrybit], outputbit);
 program = str2mat(program, cnot(selindex, carrybit));
 program = str2mat(program, cknot(2, [inputbit, carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 1) & (cbitif1==0))
 program = nnot(selindex);
 program = str2mat(program, cknot(2, [selindex carrybit], outputbit));
 program = str2mat(program, cnot(selindex, carrybit));
 program = str2mat(program, cknot(2,[inputbit, carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
 program = str2mat(program, nnot(selindex));
end;
```

Multiplexed Full Adder with an array of Enable Bits:

```
function program = muxfal(cbitif0, cbitif1,enabindex, selindex, inputbit, carrybit, outputbit)
% Edward Boyden, e@media.mit.edu
% if enabindex is 1, act like muxfa, else add zero.
% Uses cknots no higher than two greater than the number of qubits in enabindex.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = length(enabindex);
if ( (cbitif0 == 0) & (cbitif1==0))  % both cases are 0, so selindex doesn't matter
 program = cknot(2, [inputbit carrybit], outputbit);
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 1) & (cbitif1==1))  %" 1
 program = cknot(L+1, [enabindex carrybit], outputbit);
 program = str2mat(program, cknot(L, enabindex, carrybit));
 program = str2mat(program, cknot(2, [inputbit carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 0) & (cbitif1==1))   % --> add 0 or 1
 program = cknot(L+2, [enabindex selindex carrybit], outputbit);
 program = str2mat(program, cknot(L+1, [enabindex selindex], carrybit));
```

```
 program = str2mat(program, cknot(2, [inputbit, carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 1) & (cbitif1==0))
 program = nnot(selindex);
 program = str2mat(program, cknot(L+2, [enabindex selindex carrybit], outputbit));
 program = str2mat(program, cknot(L+1, [enabindex selindex], carrybit));
 program = str2mat(program, cknot(2,[inputbit, carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
 program = str2mat(program, nnot(selindex));
end;
```

Multiplexed Full Adder with an array of Enable Bits, Compiling down to C^2NOTs or lower:

```
function program = muxfall(cbitif0, cbitif1,enabindex, selindex, inputbit, carrybit, outputbit)
% Edward Boyden, e@media.mit.edu
% if enabindex is 1, act like muxfa, else add zero.
% Uses cknots no higher than two. WHILE MINIMIZING SPACE.
% Identity:
%  C[...mn],x = C[F,n],x C[...m],F C[F,n],x C[...m],F
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = length(enabindex);
if ( (cbitif0 == 0) & (cbitif1==0))  % both cases are 0, so selindex doesn't matter
 program = cknot(2, [inputbit carrybit], outputbit);
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 1) & (cbitif1==1))  %" 1
 program = cknot(L+1, [enabindex carrybit], outputbit, 2, inputbit);
 program = str2mat(program, cknot(L, enabindex, carrybit, 2, inputbit));
 program = str2mat(program, cknot(2, [inputbit carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 0) & (cbitif1==1))   % --> add 0 or 1
 program = cknot(L+2, [enabindex selindex carrybit], outputbit, 2, inputbit);
 program = str2mat(program, cknot(L+1, [enabindex selindex], carrybit, 2, inputbit));
 program = str2mat(program, cknot(2, [inputbit, carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 1) & (cbitif1==0))
 program = nnot(selindex);
 program = str2mat(program, cknot(L+2, [enabindex selindex carrybit], outputbit,2, inputbit));
 program = str2mat(program, cknot(L+1, [enabindex selindex], carrybit, 2, inputbit));
 program = str2mat(program, cknot(2,[inputbit, carrybit], outputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
 program = str2mat(program, nnot(selindex));
end;
```

Multiplexed Half Adder with an Array of Enable Bits:

```
function program = muxhal(cbitif0, cbitif1, enabindex, selindex, inputbit, carrybit)
% Edward Boyden, e@media.mit.edu
% if enabindex is 1, perform a half-added muxed half add with select to see if big or small, else
add zero. Changes the carrybit only.
% Uses cknots no higher than two greater than the number of qubits in enabindex.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = length(enabindex);
if ( (cbitif0 == 0) & (cbitif1==0))  % both cases are 0, so selindex doesn't matter
 program = cnot(inputbit, carrybit);
elseif ( (cbitif0 == 1) & (cbitif1==1))  %" 1
 program = cknot(L, [enabindex], carrybit);
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 0) & (cbitif1==1))   % --> add 0 or 1
 program = cknot(L+1, [enabindex selindex], carrybit);
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 1) & (cbitif1==0))
 program = nnot(selindex);
 program = str2mat(program, cknot(L+1, [enabindex selindex], carrybit));
 program = str2mat(program, cnot(inputbit, carrybit));
 program = str2mat(program, nnot(selindex));
end;
```

Multiplexed Half Adder with an array of Enable Bits, Compiling down to C^2NOTs or lower:

```
function program = muxhall(cbitif0, cbitif1, enabindex, selindex, inputbit, carrybit)
% Edward Boyden, e@media.mit.edu
% if enabindex is 1, perform a half-added muxed half add with select ><, else add zero.
% Uses cknots no higher than two.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SHORTCUTS
% cd 'f:\users\esb\quantumcomputing\qcc\compile';
```

```
% path(path, 'f:\users\esb\quantumcomputing\qcc');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = length(enabindex);
if ( (cbitif0 == 0) & (cbitif1==0))  % both cases are 0, so selindex doesn't matter
 program = cnot(inputbit, carrybit);
elseif ( (cbitif0 == 1) & (cbitif1==1))  %" 1
 program = cknot(L, [enabindex], carrybit, 2, inputbit);
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 0) & (cbitif1==1))   % --> add 0 or 1
 program = cknot(L+1, [enabindex selindex], carrybit, 2, inputbit);
 program = str2mat(program, cnot(inputbit, carrybit));
elseif ( (cbitif0 == 1) & (cbitif1==0))
 program = nnot(selindex);
 program = str2mat(program, cknot(L+1, [enabindex selindex], carrybit, 2, inputbit));
 program = str2mat(program, cnot(inputbit, carrybit));
 program = str2mat(program, nnot(selindex));
end;
```

Multiplexed Add:

```
function program = madd(cnumif0, cnumif1, enabindex, selindex, inputbits, outputbits,
BOOLlowisleft) % outputbits MUST be zeros
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% madd.m
% Implements adding a classical number to a quantum number, mod 2^L.
% If N is the thing we want to factor, then selindex says whether N-cnum is less than or
%  greater than B: N-cnum>b --> add cnum, else N-cnum<b --> add cnum - N + 2^L
% Enabindex must all be 1, else choose the classical addend to be zero.
% Edward Boyden, e@media.mit.edu
% INPUT
%  cnum        classical number to be added
%  indices     column vector of indices on which to operate
%  carryindex  carry qubit that you're using
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = length(outputbits);       %It's an L-bit adder: contains L-1 MUXFAs and 1 MUXHA
if (L!=length(inputbits))     %MAKE SURE OF THIS!
 program = 'Something''s wrong.';
 return;
end;
cbitsif0 = binarize(cnumif0);               % BINARIZE!
cbitsif1 = binarize(cnumif1);
cL = length(cbitsif0);
if (cL>L)
 cbitsif0 = cbitsif0((cL-L+1):cL);   % mod 2^L
else
 cbitsif0 = [zeros(L-cL) cbitsif0];
end;   % pad with zeros
cL = length(cbitsif1);
if (cL>L)
 cbitsif1 = cbitsif1((cL-L+1):cL);   % mod 2^L
else
 cbitsif1 = [zeros(L-cL) cbitsif1];
end;   % pad with zeros
cbitsif0 = reverse(cbitsif0); % reverse the cbitsifx variables to put low left
cbitsif1 = reverse(cbitsif1);
if(BOOLlowisleft)     %low IS left          %REORDER! Because we will go in small-to-large
order.
 disp 'Reversing classical bits...'
else                  %low IS right
 disp 'Reversing qubits...'
 inputbits = reverse(inputbits);
 outputbits = reverse(outputbits);
end;
program = sprintf('# Adding %d or %d + [ %s] --> [ %s], sel %d', cnumif0, cnumif1, ...
  sprintf('%d ', inputbits), sprintf('%d ', outputbits), selindex);
for i=1:(L-1)
 program = str2mat(program, muxfal(cbitsif0(i), cbitsif1(i), enabindex, selindex, inputbits(i),
outputbits(i+1), outputbits(i)));
end;
program = str2mat(program, muxhal(cbitsif0(L), cbitsif1(L), enabindex, selindex, inputbits(L),
outputbits(L)));
```

Less than:

```
function program = ltqc(cnum, qunum, scratch, selindex,BOOLlowisleft)
% scratch, selindex MUST be zero % assume low left
```

```
% Edward Boyden, e@media.mit.edu
% Compare a cnumber and a qunumber -- if qubit<cnumber, selindex = 1,
%                                else if qubit>=cnumber, selindex = 0.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = length(qunum);
S = length(scratch);
if(S!=L-1)
 disp 'Not enough scratch space...'
end;
cbits = binarize(cnum);
cL = length(cbits);
if (cL>L)
 cbits = cbits((cL-L+1):cL);  % mod 2^L
else
 cbits = [zeros(L-cL) cbits];
end;   % pad with zeros
% cbits = reverse(cbits);     % DO NOT reverse the cbits!!
if(BOOLlowisleft)
 disp 'Reversing quantum bits...'
 qunum = reverse(qunum);
else                   %low IS left
 disp 'Not reversing anything.'
end;
if (cbits(1)==0)
 program = nnot(qunum(1));
 program = str2mat(program, cnot(qunum(1), scratch(1)));
else
 program = str2mat(program, cnot(qunum(1), scratch(1)));
 program = str2mat(program, nnot(qunum(1)));
 program = str2mat(program, cnot(qunum(1), selindex));
end;
for i=2:(L-1)
 if (cbits(i)==0)
  program = str2mat(program, nnot(qunum(i)));
  program = str2mat(program, cknot(2,[scratch(i-1) qunum(i)], scratch(i)));
 else
  program = str2mat(program, cknot(2,[scratch(i-1) qunum(i)], scratch(i)));
  program = str2mat(program, nnot(qunum(i)));
  program = str2mat(program, cknot(2,[scratch(i-1) qunum(i)], selindex));
 end;
end;
if(cbits(L)==1)
 program = str2mat(program, nnot(qunum(L)));
 program = str2mat(program, cknot(2, [scratch(L-1), qunum(L)], selindex));
end;
```

Auxiliary programs:

```
function s=binarize(d,n)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% like DEC2BIN Convert decimal integer to a binary string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n = 1;
[f,e]=log2(max(d)); % How many digits do we need to represent the numbers?
s=rem(floor(d*pow2(1-max(n,e):0)),2);

function x=decimize(v)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% like BIN2DEC Convert binary array to decimal integer.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[m,n] = size(v);
twos = pow2(n-1:-1:0);
x = sum(v .* twos(ones(m,1),:),2);

function revarray = reverse(array)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
l = length(array);
revarray = array(l+1-(1:l));
```

Appendix B: Simulation Code, with Examples

This code helps simulate NMR quantum computing operations, using the product operator formalism. Several examples are given at the end. The core procedure, which builds up global simulations from two-qubit operations, is the function `krqn`, which does a generalized tensor product so as to embed a simple, few-qubit operation into a full density-matrix operator.

At the time of this writing, there is no Matlab parser to take the code generated by the **Appendix A** compiler and convert them to a computational representation that would be usable in the following architecture.

Rotation by θ around the *x*-axis:

```
function matrix = X(theta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% X(theta) returns the matrix
%  [ cos(theta/2) i * sin(theta/2) ]
%  [                               ]
%  [ i * sin(theta/2) cos(theta/2) ]
%
matrix = [ cos(theta/2) i*sin(theta/2); i*sin(theta/2) cos(theta/2) ];
```

Rotation by θ around the *y*-axis:

```
function matrix = Y(theta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Y(theta) returns the matrix
%  [ cos(theta/2)     sin(theta/2) ]
%  [                               ]
%  [ -sin(theta/2)    cos(theta/2) ]
%
matrix = [ cos(theta/2) sin(theta/2); -sin(theta/2) cos(theta/2) ];
```

Rotation by θ around the *z*-axis:

```
function matrix = Z(theta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Z(theta) returns the matrix
%  [ exp(i*theta/2)           0    ]
%  [                               ]
%  [ 0           exp(-i*theta/2)   ]
%
matrix = [ exp(i*theta/2) 0; 0 exp(-i*theta/2)];
```

J-coupling between two qubits

```
function matrix = ZZ(theta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ZZ(theta) returns the matrix exp(-i*theta/2 * 2*Iz*Sz)
theta2 = theta;% /2; % remove this if it doesn't work
matrix = diag([exp(i*theta2/2) exp(-i*theta2/2) exp(-i*theta2/2) exp(i*theta2/2)]);
```

C^kNOT:

```
function m = cknot(k, n)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cknot.m
% Matrix of cknot.
% Edward Boyden, e@media.mit.edu
% INPUT
%  k    order of cknot
%  n    index to be changed
m = eye(2^k, 2^k);
% must swap the |11.... 0 ... 1> and |11.... 1 ... 1> indices
i = 2^k-2^(k-n);
j = 2^k;
m(i,j) = 1;
m(j,i) = 1;
```

```
m(i,i) = 0;
m(j,j) = 0;
```

Conditional operator:

```
function bigop = condit(numQBs, indices, outdices, littleop)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% condit.m
% Implements conditional form, littleop_[indices],index
% Edward Boyden, e@media.mit.edu
% INPUT
%  numQBs   the number of qubits in the whole system
%  indices  the qubit numbers that littleop conditions on
%  outdices the qubits that littleop works on
%  littleop a few-qubit operator
M = length(outdices);
L = length(indices)+M;
twototheL = 2^L;
twototheM = 2^M;
% assemble an operator that operates conditionally
% It's just an identity matrix with the lower right corner set to the littleop.
% the krqn procedure will automatically take care of the reordering!!!!!
operat = eye(twototheL);
operat( (twototheL - twototheM + 1):twototheL, (twototheL - twototheM + 1):twototheL) = littleop;
bigop = krqn(numQBs, [indices outdices], operat);
```

Kronecker tensor product (for lexicographically-ordered qubit operations):

```
function bigop = krqn(numQBs, indices, littleop)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% krqn.m
% Implements quantum kronecker tensor product
% Edward Boyden, e@media.mit.edu
% INPUT
%  numQBs   the number of qubits in the whole system
%  indices  the qubit numbers that littleop works on
%           length(indices) = size(littleop)
%  littleop a few-qubit operator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
i=0;
L = length(indices);
bigsize = 2^numQBs;
littlesize = 2^L;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(max(indices)>numQBs) disp('Too few QBits!'); return; end;
bigop = [1];
% easy case
if ((L==1)&(indices(1)==numQBs))
 bigop = kron(eye(bigsize/2, bigsize/2), littleop);
 return;
% slightly harder case: still one qubit littleop
elseif (L==1)
 for i=1:(indices(1)-1),
  bigop = kron(bigop, eye(2,2));
 end;
 bigop = kron(bigop, littleop);
 for i=(indices(1)+1):numQBs,
  bigop = kron(bigop, eye(2,2));
 end;
 return;
% more than one qubit littleop, but all the indices are consecutive
elseif((sum(indices==[indices(1):(indices(1)+L-1)]))==L) %gotta love matlab
 for i=1:(indices(1)-1),
  bigop = kron(bigop, eye(2,2));
 end;
 bigop = kron(bigop,littleop);
 for i=(indices(1)+L):numQBs,
  bigop = kron(bigop, eye(2,2));
 end;
%return;
% arbitrary qubit littleop, arbitrary indices
else
 bigop = eye(2^numQBs, 2^numQBs);
 ind2=2.^(numQBs-indices);
 for i=1:(2^(numQBs))
  for j = 1:(2^(numQBs))
```

```
   ii=i-1; jj=j-1;
   % find ii,jj's 2^a and 2^b components.
   iix = bitand(ii,ind2);
   jjx = bitand(jj,ind2);
   if ((ii-sum(iix))==(jj-sum(jjx)))
    iix = (iix>0).*2.^((L-1):-1:0);
    jjx = (jjx>0).*2.^((L-1):-1:0);
    bigop(i,j) = littleop(1+sum(iix), 1+sum(jjx));
   else
    bigop(i,j) = 0;
   end;
  end;
 end;
end;
```

Sample code: declarations:

```
sigma1 = [0 1; 1 0]
sigma2 = [0 -i; i 0]
sigma3 = [1 0; 0 -1]
Ix = 1/2*sigma1
Iy = 1/2*sigma2
Iz = 1/2*sigma3
H = 1/sqrt(2) * [1 1; 1 -1]
U = 1/sqrt(2) * [1 1; -1 1]
V = [1 0; 0 -1]
P = [1 0; 0 1]
CV = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 -1]
NOT = [0 1; 1 0]
CNOT = condit(2,1,2,NOT)
CNOT_2 = condit(2,2,1,NOT)
C0NOT = [0 1 0 0 ; 1 0 0 0 ; 0 0 1 0; 0 0 0 1]
swap = CNOT * CNOT_2 * CNOT
```

Sample code: simple examples:

```
krqn(3, [2 3], CNOT)      % should return C2NOT
condit(3, [1 2], 3, NOT)  % also returns C2NOT
condit(3, 1, [2 3], CNOT) % also returns C2NOT
```

Decomposing C^kNOT:

```
condit(7, [1 2 3 4 6], 7, NOT);     % same thing as a1*a2*a3*a4, which are defined as:
a1= condit(7, [1 2 3 4], 5, NOT);
a2= condit(7, [5 6], 7, NOT);
a3= condit(7, [1 2 3 4], 5, NOT);
a4= condit(7, [5 6], 7, NOT);
% Compare to cknot(5,[1 2 3 4 6], 7, 4, 5), in the compiling section.
```

Decomposing the Toffoli gate:

```
b1 = krqn(3, [2 3], condit(2,[1],[2],1/sqrt(2)*(sigma2+sigma3)) )     % same as "Va,b"
b2 = krqn(3, [1 3], condit(2,[1],[2],sigma2) )                        % same as Ua,b
% b1*b2*b1*b2 is conditional phase shift by pi/2 – let's erase the phase shift that arises
b3 = condit(3, [1 2], [3], exp(i*pi/2)*P)                             % same as CPab,(3pi/2)
b4 = b3*b1*b2*b1*b2                                                   % equals C2NOTab,c
```

Implementing Hadamard in NMR operations:

```
c1 = pi/4;
c2 = [cos(c1/2), sin(c1/2); -sin(c1/2) cos(c1/2)]      % Y(pi/4)
c3 = pi;
c4 = [cos(c3/2), i*sin(c3/2); i*sin(c3/2) cos(c3/2)]   % X(pi)
c5 = -pi/4;
c6 = [cos(c5/2), sin(c5/2); -sin(c5/2) cos(c5/2)]      % Y(pi/4)
c2*c4*c6                                               % --> H
```

The other Hadamard Gate:

```
c1 = pi/2;
c2 = [cos(c1/2), sin(c1/2); -sin(c1/2) cos(c1/2)]      % = Y(pi/2) --> U
```

An abbreviated CNOT gate and EPR gate (Chu98a):

```
d1 =  krqn(2, 2, Y(pi/2)) * ZZ(pi/2) * krqn(2, 2, X(pi/2))
Neilsmatrix = [ (-1)^(1/4) 0 0 0; 0 -(-1)^(3/4) 0 0 ; 0 0 0 (-1)^(1/4); 0 0 (-1)^(3/4) 0]
sum(sum(abs(d1-Neilsmatrix)))                                          % = 0, of course
krqn(2,2,Y(pi/2)) * ZZ(pi/2) * krqn(2,1,X(pi/2)) * krqn(2,2,X(pi/2))    % the EPR matrix
```

Two ways to get *z*-rotations from *x*-rotations and *y*-rotations only:

```
thetaX = pi/2
thetaY = .37
Xtheta = [ cos(thetaX/2) i * sin(thetaX/2); i*sin(thetaX/2) cos(thetaX/2)]
Ytheta = [ cos(thetaY/2) sin(thetaY/2); -sin(thetaY/2) cos(thetaY/2) ]
inv(Xtheta) * Ytheta * Xtheta
thetaX = .37
thetaY = pi/2
Xtheta = [ cos(thetaX/2) i * sin(thetaX/2); i*sin(thetaX/2) cos(thetaX/2)]
Ytheta = [ cos(thetaY/2) sin(thetaY/2); -sin(thetaY/2) cos(thetaY/2) ]
Ytheta * Xtheta * inv(Ytheta)
```

Ways to implement the full CNOT:

```
CNOT12 = krqn(2,2,Y(-pi/2)) * krqn(2,1,Z(-pi/2)) * krqn(2,2,Z(-pi/2)) * ZZ(pi/2) *
krqn(2,2,Y(pi/2))
CNOT12 = krqn(2,1,Y(pi/2)*X(-pi/2)*Y(-pi/2)) * krqn(2,2,X(-pi/2)*Y(-pi/2)) * ZZ(pi/2) *
         krqn(2,2,Y(pi/2))    % only one J-coupling time
CNOT12 = krqn(2,2,X(-pi/2)*Y(-pi/2)) * krqn(2,1,Y(pi/2)*X(-pi/2)*Y(-pi/2)) * ZZ(pi/2) *
         krqn(2,2,Y(pi/2))
CNOT21 = krqn(2,1,X(-pi/2)*Y(-pi/2)) * krqn(2,2,Y(pi/2)*X(-pi/2)*Y(-pi/2)) * ZZ(pi/2) *
         krqn(2,1,Y(pi/2))
CNOT12 = krqn(2,2,Y(pi/4)*X(pi)*Y(-pi/4)) * diag([1 1 1 -1]) * krqn(2,2,Y(pi/4)*X(pi)*Y(-pi/4))
% can we make the conditional phase shift any shorter?
```

Grover's algorithm, as implemented in (Chu98b):

```
% Three ways to make a Walsh-Hadamard transform (up to phase):
% W1 = W2 = W3 = -H
W1 = krqn(2,1,X(pi)*Y(-pi/2)) * krqn(2,2,X(pi)*Y(-pi/2))
W2 = krqn(2,1,Y(pi/4)*X(pi)*Y(-pi/4)) * krqn(2,2,Y(pi/4)*X(pi)*Y(-pi/4))
W3 = krqn(2,1,Y(pi/2)*X(pi)) * krqn(2,2,Y(pi/2)*X(pi))

% our function – let's say we want the fourth term
C1 = diag([1 1 1 -1])
% written in a practical form: C2 is C1*exp(-i*pi/2)
C2 = krqn(2,1,Y(pi/2)*X(-pi/2)*Y(-pi/2)) * krqn(2,2,Y(pi/2)*X(-pi/2)*Y(-pi/2)) * ZZ(pi/2)

% next invert about the average, D = WPW where P = diag([1 -1 -1 -1])
P1 = diag([1 -1 -1 -1])
% written in a practical form: P2 = P1 * exp(i*3*pi/4)
P2 = krqn(2,1,Y(pi/2)*X(pi/2)*Y(-pi/2)) * krqn(2,2, Y(pi/2)*X(pi/2)*Y(-pi/2)) * ZZ(pi/2)

% the operators themselves
D1 = W3 * P2 * W3
U1 = D1 * C2
% another way to make the U-operator, still for the case of the function C1 given above
U2 = krqn(2,1,X(-pi/2)*Y(-pi/2)) * krqn(2,2,X(-pi/2)*Y(-pi/2)) * ZZ(pi/2) * krqn(2,1, X(pi/2)*
     Y(-pi/2)) * krqn(2,2,X(pi/2)*Y(-pi/2)) * ZZ(pi/2)

% SIMULATION
U1 * W3 * [ 1 0 0 0]' % results in [0 0 0 1]', giving the correct answer

% for other functions – one that recognizes the first database entry, say
U5 = krqn(2,1,X(-pi/2)*Y(-pi/2)) * krqn(2,2,X(-pi/2)*Y(-pi/2)) * ZZ(pi/2) * krqn(2,1, X(-
pi/2)*Y(-pi/2)) * krqn(2,2,X(-pi/2)*Y(-pi/2)) * ZZ(pi/2)
U5 * W3 * [1 0 0 0]'    % [-i 0 0 0]

% for other functions – the second term, say
U3 = krqn(2,1,X(-pi/2)*Y(-pi/2)) * krqn(2,2,X(-pi/2)*Y(-pi/2)) * ZZ(pi/2) * krqn(2,1, X(pi/2)*
     Y(-pi/2)) * krqn(2,2,X(-pi/2)*Y(-pi/2)) * ZZ(pi/2)
U3 * W3 * [1 0 0 0]'  % returns [0 -1 0 0]'

% for other functions – the third term, say
U4 = krqn(2,1,X(-pi/2)*Y(-pi/2)) * krqn(2,2,X(-pi/2)*Y(-pi/2)) * ZZ(pi/2) * krqn(2,1, X(-
pi/2)*Y(-pi/2)) * krqn(2,2,X(pi/2)*Y(-pi/2)) * ZZ(pi/2)
U4 * W3 * [1 0 0 0]'    % returns [ 0 0 -1 0]'
```

Temporal averaging:

```
% start with typical deviation density matrix
rho = diag([1 .6 -.6 -1]);
% first experiment - use unmodified result
Pr1 = eye(4)
% second experiment - permute 01 -> 10 -> 11 -> 01, using Pr2
```

```
Pr2 = [1 0 0 0; 0 0 1 0; 0 0 0 1; 0 1 0 0]
% realistic way to make Pr2:
Pr2 = CNOT12 * SWAP
% third experiment - inverse the permutation, using transpose of the previous operation
Pr3 = Pr2'
% realistic way to make Pr3:
Pr3 = SWAP * CNOT12
Pr3 = CNOT12 * CNOT21
Pr1 * rho * Pr1' + Pr2 * rho * Pr2' + Pr3 * rho * Pr3'
% → the final result is diag([3 -1 -1 -1]), an effective pure state for bulk quantum
computation!
```

Appendix C: nanoTag and nanoNMR

nanoTag code

The tag reader is described in some detail in **VI.2.i.**

Here is the Verilog code that drives the tagreader. Note that the frequency word is 32 bits long, as is the phase accumulator, but we only take the top 8 bits for conversion to the sinewave amplitude. Only part of the lookup table is shown, for brevity.

```
//DDS 1.0, eboyden3
module kam(CLOCK, DACword, RAM_CS, RAM_OE, INTEL_RST, DAC_LATCH, ADCword, ADCencode,LED);

//Declarations
input CLOCK;
output [7:0] DACword;
reg [7:0] DACword;
output RAM_CS, RAM_OE, INTEL_RST, DAC_LATCH;
reg DAC_LATCH;
input [11:0] ADCword;
output ADCencode;
reg ADCencode;
output [6:0] LED;
reg [6:0]LED;

reg [11:0] signal;
reg [31:0] signal_accum;
reg CLOCK_DIV1048576;
reg CLOCK_DIV65536;
reg CLOCK_DIV64;
reg CLOCK_DIV32;
reg CLOCK_DIV16;

reg [19:0] CLOCK_INTERNAL_COUNTER; //20-bit counter
reg [31:0] freq_word;
reg [31:0] freq_freq_word;
reg [31:0] phas_accum;

//disable the RAM and 8031 by pulling CS, OE, and RST high.
assign RAM_CS = 1;
assign RAM_OE = 1;
assign INTEL_RST = 1;

//Divide the clock down
always begin
 @(posedge CLOCK)
 CLOCK_DIV1048576 = 0;
 CLOCK_DIV64 = 0;
 CLOCK_DIV32 = 0;
 CLOCK_DIV16 = 0;
 CLOCK_INTERNAL_COUNTER = 0;

 forever begin
  @(posedge CLOCK)
  CLOCK_INTERNAL_COUNTER = CLOCK_INTERNAL_COUNTER + 1;
  CLOCK_DIV1048576 = CLOCK_INTERNAL_COUNTER[19]; //divide by 2^20 = 1048576
  CLOCK_DIV65536 = CLOCK_INTERNAL_COUNTER[15]; //divide by 65536
  CLOCK_DIV64 = CLOCK_INTERNAL_COUNTER[5]; //e.g. divide by 64
  CLOCK_DIV32 = CLOCK_INTERNAL_COUNTER[4]; //e.g., divide by 32
  CLOCK_DIV16 = CLOCK_INTERNAL_COUNTER[3]; //e.g., divide by 16
 end
end

always begin //use only one clock per always block, or get multiple wait states
 forever begin
  @(posedge CLOCK_DIV32); //posedge CLOCK_DIV32
  DAC_LATCH = 0;          //happens __|
  @(posedge CLOCK_DIV32); //posedge CLOCK_DIV32
```

```
  DAC_LATCH = 1;           //happens __|--|__|
 end
end

always begin
 @(posedge CLOCK_DIV65536)
 //15D867C = 1 kHz, 51EB8510 = 60 kHz, 6D3A06D4 = 80 kHz
 //(freq_word_max - freq_word_min) / 2^32 * (65536 / 12e6) * (12e6/64) = 322.21875 Hz per step
 //322.21875 Hz per step = 70A07F, using freq_word/2^32*12e6/64 = frequency
 freq_freq_word = 32'h70A07F;   //to sweep in one second, do it in 12000000 / 65536 = 183.1 steps
 freq_word = 32'h15D867C;       //happens __|

 forever begin
  @(posedge CLOCK_DIV65536)
   freq_word = freq_word + freq_freq_word;
 end
end

always begin
 @(posedge CLOCK_DIV64)
 phas_accum = 32'h0;
 DACword = 8'd127;

 forever begin        //at each clockstep, in parallel, increment the phase accumulator
  @(posedge CLOCK_DIV64)   //posedge CLOCK_DIV32 - one sample every 187500
phas_accum = phas_accum + freq_word; //happens __|--|__|
  case (phas_accum[31:24])
    8'h0:DACword = 8'h7f;
    8'h1:DACword = 8'h82;
    8'h2:DACword = 8'h85;
    8'h3:DACword = 8'h88;
    8'h4:DACword = 8'h8b;
    8'h5:DACword = 8'h8e;
    8'h6:DACword = 8'h91;
    8'h7:DACword = 8'h94;
    8'h8:DACword = 8'h97;
    8'h9:DACword = 8'h9a;
    8'ha:DACword = 8'h9d;
    .
    . // this part is generated by a C program, and is just a 8-to-8 phase lookup table
    . // see below for the code that generates this lookup table
    .
    8'hf9:DACword = 8'h69;
    8'hfa:DACword = 8'h6c;
    8'hfb:DACword = 8'h6f;
    8'hfc:DACword = 8'h72;
    8'hfd:DACword = 8'h75;
    8'hfe:DACword = 8'h78;
    8'hff:DACword = 8'h7b;
  endcase
 end //end forever
end //end always

//now we need the concept of a sweep.
//whenever the frequency is incremented, display detect_accum (top four bits) and set
detect_accum to 0
//whenever a 12 bit read comes in, and a 8 bit goes out, multiply them and add to the accumulator
always begin
 @(posedge CLOCK_DIV16)
 signal = 0;
 ADCencode = 1;
 @(posedge CLOCK_DIV16)
 ADCencode = 0;

 forever begin        //at each clockstep, in parallel, increment the phase accumulator
  signal = ADCword;   //grab whatever was made at the last time
  @(posedge CLOCK_DIV16)   //posedge CLOCK_DIV16 - 750000 samples per second
  ADCencode = 1;
  @(posedge CLOCK_DIV16) //so there ends up being one per
  ADCencode = 0;
 end
end //end always

always begin
 @(posedge CLOCK_DIV64)
```

```
 signal_accum = 268435456; //1024 zeroes, zero = 262144

 //each frequency is used for 65536/CLOCK seconds - thus CLOCK_64 ticks 1024 times
 //for each frequency. Therefore each tick needs to be 1023/1024 * old + 1/1024 * new
 // = old - 1/1024 * old + 1/1024 * new = old - old >> 10 + new >> 10
 // scale up to 1024 times that, so that we don't lose any bits
 // = old - old >> 10 + new //therefore each time, we discard about 1/1000 and add another,
 // which reaches a steady state when new = old >> 10, or when old ~ 1000 times new, which is
 // what we want.
 // 12 bits per signal, 8 for DAC, 2^10 entries -> 30 bits total -- make it an even 32
 forever begin
  @(posedge CLOCK_DIV64)
  //2048 = zero for signal, 128 = zero for DACword
  signal_accum = signal_accum - (signal_accum >> 10) + (signal * DACword -
                 (DACword << 11) - (signal << 7) + 262144);
 end //end forever
end //end always

always begin //update the display 11.4 times a second, with CLOCK_DIV1048576
 @(posedge CLOCK_DIV1048576) begin
    case (signal_accum[30:27])
        4'b0001 : LED = 7'b1111001;   //1
        4'b0010 : LED = 7'b0100100;   //2
        4'b0011 : LED = 7'b0110000;   //3
        4'b0100 : LED = 7'b0011001;   //4
        4'b0101 : LED = 7'b0010010;   //5
        4'b0110 : LED = 7'b0000010;   //6
        4'b0111 : LED = 7'b1111000;   //7
        4'b1000 : LED = 7'b0000000;   //8
        4'b1001 : LED = 7'b0010000;   //9
        4'b1010 : LED = 7'b0001000;   //A
        4'b1011 : LED = 7'b0000011;   //b
        4'b1100 : LED = 7'b1000110;   //C
        4'b1101 : LED = 7'b0100001;   //d
        4'b1110 : LED = 7'b0000110;   //E
        4'b1111 : LED = 7'b0001110;   //F
        default : LED = 7'b1000000;   //0
    endcase
//  signal_accum = 0;
   end //end @
end //end always

endmodule
```

Below is the code that generates the sinusoidal lookup table.

```
/* ROMGEN.c rom generator for FPGA (Verilog code) E. Boyden */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.141592653589793

int main() {
        int i;
        float val;
        int inbits,innum,outbits,outnum;
        FILE*romfile;
        char romfilename[]="output_verilog_rom.txt";
        inbits = 8;
        innum = pow(2,inbits);
        outbits = 8;
        outnum = pow(2,outbits-1)-1;
        romfile = fopen(romfilename,"w");
        fprintf(romfile," case (phas_accum[31:24])\n");
        for(i=0;i<innum;i++) {
                val = sin(2*PI*i/innum);
                val *= outnum;
                val += outnum;
                fprintf(romfile, "    %d'h%x:DACword = %d'h%x;\n", inbits, i, outbits,(int)val);
        }
        fprintf(romfile, " endcase\n");
        fclose(romfile);
}
```

nanoNMR transmitter schematics

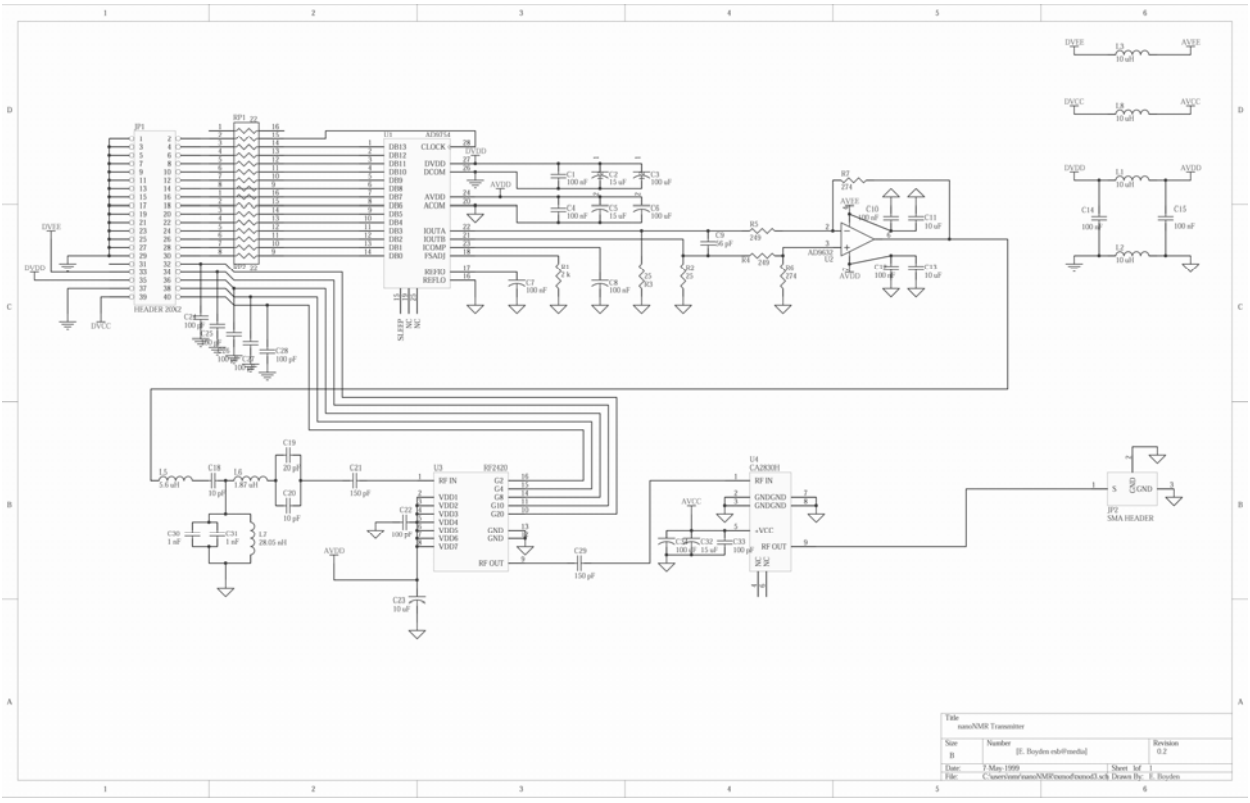


Figure 11: nanoNMR transmitter board schematic

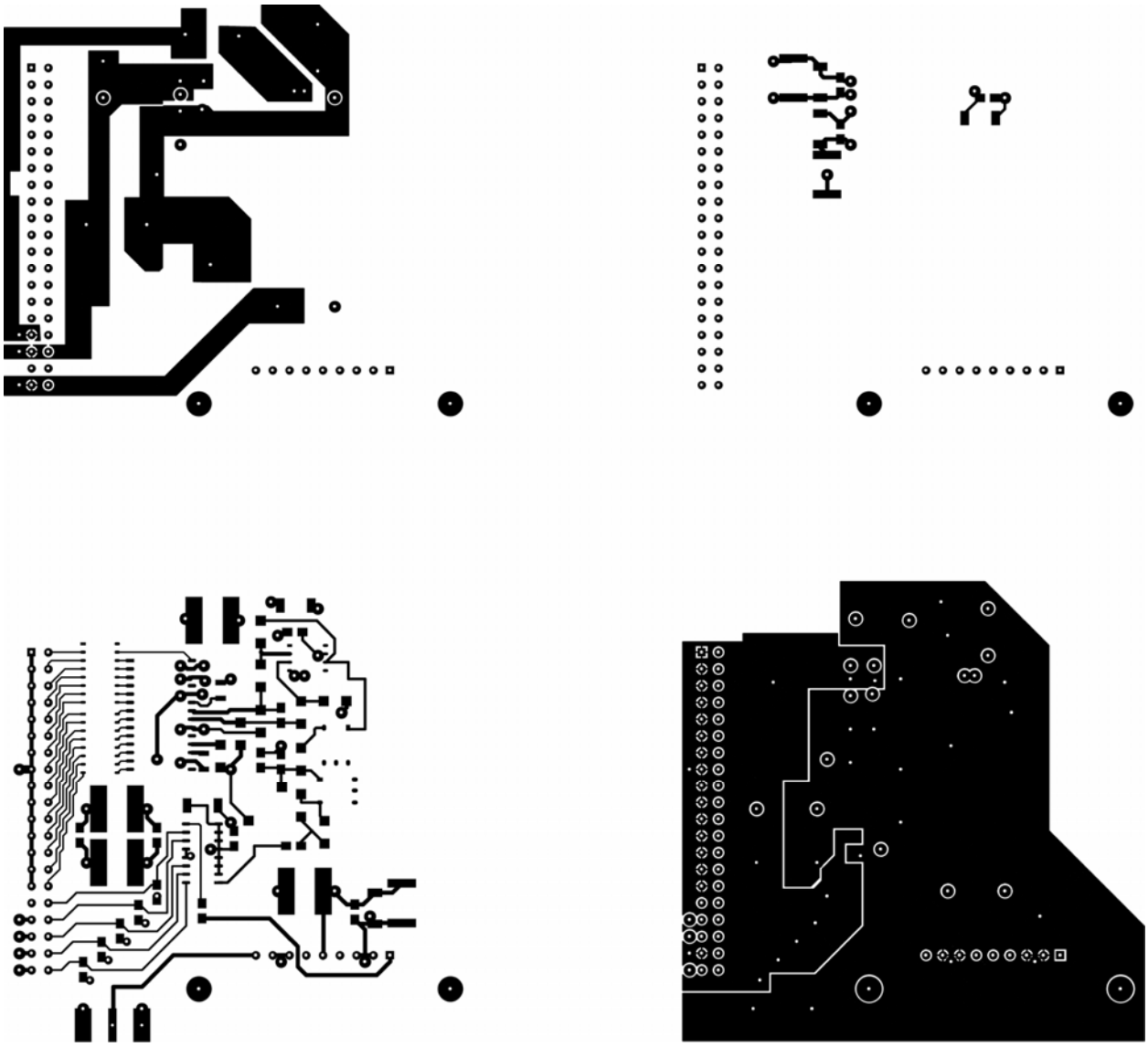


Figure 12: PCB layout of nanoNMR transmitter

Appendix D: The *q* language and the ROS scripting language

The following lines describe the commands of the *q* language and the ROS scripting language, along with the relevant system parameters and modes of behavior. *q* and ROS should be thought of as *experiment description languages*.

Also included in this appendix is an extended example detailing the 27 programs needed to perform Grover's algorithm with temporal averaging and state tomography. **Appendix E** contains some of the code from the actual QOS program, and **Appendix F** contains some of the code from ROS, for reference in understanding the following *q* programs and ROS scripts, as well as understanding the operation of the quantum computer.

Delimiters between symbol strings: ,;()[]{}' \t\n\r

> **D_i{*target*}** denotes a symbol string that must be found within the *target* string (e.g., if the target string is 'hemisphere' then 'hemisp' will be just fine, but 'nothemisphere' or 'hemispheree' or 'azxvtt' will cause the parser to assume a default state).

Delimiters between name strings: '\n()

> **S_x[*n*]** denotes that the first *n* characters of the name string are directly copied into a variable in memory, for later use.

These delimiters are designed to match that of the Matlab parser, so that they can be interpreted as Matlab commands. Due to the very dynamic state of the *q* language, however, it was never deemed worthwhile to actually write Matlab simulation programs for these commands.

The *q* commands

Qbegin()
Qend()
Do nothing. Used to frame the program.

Qname(S_x[8])
Qversion(S_x[8])
Qsample(S_x[8])
For human debugging and record-keeping. These are used to label files and appear in the QOS logging system amongst many other parameters of each QOS program execution.

Qdecouplingstyle(D_i{'minimal', 'maximal'})
Indicates what kind of decoupling is used, during waiting periods. Only `minimal` is currently supported, meaning no decoupling, since we have only two spins in our current system. Default state is minimal.

Qmode(D_i{'verbose'},
Print (and log) all the behaviors of QOS. Default state is no printing.

D_i{'simulateonly'},
Simulate actions only. Don't send any commands to the hardware. Default state is not simulating.

D_i{'continuous'},
Continuously stream the waveform from the HPE1445 and acquire a stream of data from the HPE1437. Does not save any data, and is not interruptable without losing the state of the system

– this mode is only useful for debugging 1445 outputs on the oscilloscope. Default state is to send, and acquire, in discrete events.

D_i`{'triggered'},`

If enabled, the 1437 acquisition must be triggered by an external pulse (e.g., from the Hitachi microcontroller). Default state is instantaneous triggering.

D_i`{'armed'},`

If enabled, the 1445 waveform generation must be armed by an external pulse (from the Hitachi). Default state is instantaneous arming.

D_i`{'instantreplay', 'notinstantreplay'},`

Mode 'instantreplay' means to keep repeating the program that follows – all parts of the waveform, triggering, acquisition, and so on. This mode is to be contrasted with 'continuous', which simply causes the HP modules to loop forever, with no switching, triggering, etc. 'notinstantreplay' means to terminate after one execution of the *q* program. Default state is to give a prompt, at which the user chooses one of these two states.

D_i`{'specifytesla', 'specifyduration'},`

Mode 'specifytesla' means to specify the exact strength of the magnetic field induced by the RF pulse – a difficult thing to do. 'specifyduration' means to specify the length of a $\pi/2$ pulse. Default state is to display an error message and exit QOS – that is, one of these two choices must be specified in the *q* program.

D_i`{'specifyJtimes', 'specifyJcouplings'} )`

Mode 'specifyJtimes' means to specify the desired J-coupling times between each pair of spins for each individual wait command, while 'specifyJcouplings' means to specify all the J-coupling strengths in a matrix, then pass angles into each wait command. Default state is to display an error message and exit QOS.

`Qmodules(`D_i{'1437'},

'1437' indicates that the 1437 is to be active. Default state is to not send commands to the 1437.

D_i`{'1445'},`

'1445' indicates that the 1445 is to be active. Default state is to not send commands to the 1445.

D_i`{'hitachi'},`

'hitachi' indicates that the Hitachi microcontroller is to be active. Default state is to not send commands to the Hitachi microcontroller.

D_i`{'shim'}`

'shim' indicates that the shimming system is to be active. Default state is to not send commands to the shimming system.

(Note: If any module is specified as used, and is broken or off, then QOS has undefined behavior. If any module is specified as omitted, then its physical performance has no bearing on the operation of QOS. Therefore the `Qmodules` command is good for debugging the behavior of specific hardware modules.)

`Qrfsettings( int, int)`

Two parameters, each comprising 10 bits, set the level of the RF2420 digital attenuators for the ^{13}C and ^{1}H sections of the transmitter board.

`Qhack( float,`

A delay which accounts for the fact that 1445 signal generation doesn't begin immediately when armed. The Hitachi waits this amount of time (in seconds) before starting its parallel sequence of gating commands.

`float,`

This parameter tells the Hitachi how wide the framing of each pulse should be. The idea is that

the gating commands should close the switches before each pulse begins, and open the switches after each one ends, so that no transmit signal is lost – one can think of this as a margin-of-error parameter. The value is specified in seconds.

float)

A ringdown delay. This tells the pin diode switch and 1437 not to switch into acquisition mode until the coil has had a chance to radiate away any residual transmitter energy. The value is specified in seconds.

Qdigitize(float,

The center frequency at which the 1437 reports its data, expressed as a fraction of 20 MHz – e.g., using 0.1 would make the center frequency 2 MHz. The 1437 digitally performs the frequency shift, returning a complex time series.

int,

An integer *s* specifying the signal bandwidth of the 1437. The bandwidth is roughly $f_s/(2.56 \times 2^s)$, where f_s is 20 MHz (or 10 MHz, if decimation is on). The sample rate is $f_s/2^{s-1}$, except when *s* is zero, in which case it is f_s. The parameter *s* can be between 0 and 24.

int,

An integer *v* specifying the full-scale voltage range of the 1437, which is $\pm 0.02 \times 2^v$.

int,

The blocksize for the 1437. This specifies the number of bytes that an acquisition takes up in memory.

D_i{'decimate'})

Whether to decimate or not. This affects the parameter f_s. Default is no decimation.

Qnumqbs(int)

Number of qubits *numqbs* in the current sample.

Qspecies(int,

The identity of the current qubit, expressed as a decimal integer. Can take the values 1, 2, . . . *numqbs*.

S_x[8],

The name of the current species. For human interpretation of the program.

float,

Gyromagnetic ratio, in rad T^{-1} s^{-1}.

float,

Larmor frequency in Hz, *Larmor*.

float,

Frequency *CWfreq* of the CW generator which is feeding into the mixer. The 1445 therefore synthesizes pulses for this particular species at the frequency *Larmor – CWfreq*.

float,

Sensitivity, relative to ^{1}H. Not used in this implementation of QOS.

float)

If `Qmode` argument 7 is 'specifyduration', this is the length of a $\pi/2$ pulse for the current species. If `Qmode` argument 7 is 'specifyTesla', then this parameter is the radiofrequency magnetic field per unit voltage from the 1445. Using 'specifyTesla' is not recommended due to the difficulty of estimating this dependence.

Qdatatype(D_i{'complex', 'real'},

Default is real data, from the 1437.

int,

32 or 16 bits. Default is 16.

D_i{'powerspectrum', 'fft', 'timedomain'})

Default is FFT.

Qshims(int,

The number of shim coils.

float, float, float, . . . }

The current that passes through each one. The sign determines the direction of the current, and therefore the orientation of the magnetic field. In the current implementation of the shim hardware, this direction is controlled by the state of an electromechanical relay.

Qx(. . .
Qy(int,

The species to be addressed.

float,

The angle that the spin is to be tipped about the *x*- or *y*-axis.

D_i{‘square’, ‘gaussian’, ‘halfgaussian’, ‘arbfreq’}

The type of pulse.

→ For a ‘square’ pulse, there are no further parameters. The Qx or Qy pulse simply tips the indicated species about the *x* or *y* axis, respectively, by the indicated angle.

→ For a ‘gaussian’ pulse, there are three more parameters, detailing the Gaussian envelope which is applied (in time-domain) to the pulse:

float,

The time at which the Gaussian envelope is at its maximum, expressed as a fraction of the pulse length. For example, if the pulse is 0.012 seconds long and this parameter is 0.4, then the envelope will be at its maximum 0.0048 seconds into the pulse.

float,

The width (standard deviation) of the pulse, expressed as a fraction of the pulse length. The width can be much greater than 1, if a gentle envelope is desired.

float)

The amplitude of the Gaussian envelope.

→For a ‘halfgaussian’ pulse, there are three more parameters, detailing the half-Gaussian envelope which is applied (in time-domain) to the pulse:

float

The time at which the Gaussian envelope is at its maximum, expressed as a fraction of the pulse length. For example, if the pulse is 0.012 seconds long and this parameter is 0.4, then the envelope will be at its maximum 0.0048 seconds into the pulse. The sign of this value determines whether the half-Gaussian falls of to the right (e.g., the envelope is zero to the left of the maximum) or the left (e.g., the envelope is zero to the right of the maximum).

float

The width (standard deviation) of the pulse, expressed as a fraction of the pulse length. The width can be much greater than 1, if a gentle envelope is desired.

float)

The amplitude of the Gaussian envelope.

→ For an ‘arbfreq’ pulse, there are three more parameters, detailing the modulation which is to be applied to the pulse. The pulse is multiplied by $A \sin(\omega_{ARB} t + \phi)$.

float,

ϕ, the phase of the modulation.

float,

$f_{ARB} = \omega_{ARB}/(2\pi)$, the frequency of the modulation. Note that the 1445 will generate the modulation signal at the frequency ($CWfreq - f_{ARB}$), since the digitally-synthesized 1445 signal will be mixed with the CW frequency in the analog hardware. Also note that due to the

bandpass filters in the analog hardware, f_{ARB} cannot be too far off from the Larmor frequency.

float)

A, the amplitude of the modulation.

Qp(int,

The species to be addressed. The `Qp` pulse is an arbitrary sinusoid, $B \sin(\omega_P t + \Phi)$, applied to the indicated species.

float,

The duration of the `Qp` pulse, in Larmor periods of the indicated spin.

float,

The phase of the pulse, Φ.

float,

The frequency of the pulse, $f_P = \omega_P/(2\pi)$.

float,

The amplitude of the waveform, *B*.

D_i{'square', 'gaussian', 'halfgaussian', 'arbfreq'},

The type of pulse.

→ For a 'square' pulse, there are no further parameters. The `Qp` pulse simply tips the indicated species about the indicated arbitrary axis for the indicated time.

→ For a 'gaussian' pulse, there are three more parameters, detailing the Gaussian envelope which is applied (in time-domain) to the pulse:

float,

The time at which the Gaussian envelope is at its maximum, expressed as a fraction of the pulse length. For example, if the pulse is 0.012 seconds long and this parameter is 0.4, then the envelope will be at its maximum 0.0048 seconds into the pulse.

float,

The width (standard deviation) of the pulse, expressed as a fraction of the pulse length. The width can be much greater than 1, if a gentle envelope is desired.

float)

The amplitude of the Gaussian envelope.

→For a 'halfgaussian' pulse, there are three more parameters, detailing the half-Gaussian envelope which is applied (in time-domain) to the pulse:

float

The time at which the Gaussian envelope is at its maximum, expressed as a fraction of the pulse length. For example, if the pulse is 0.012 seconds long and this parameter is 0.4, then the envelope will be at its maximum 0.0048 seconds into the pulse. The sign of this value determines whether the half-Gaussian falls of to the right (e.g., the envelope is zero to the left of the maximum) or the left (e.g., the envelope is zero to the right of the maximum).

float

The width (standard deviation) of the pulse, expressed as a fraction of the pulse length. The width can be much greater than 1, if a gentle envelope is desired.

float)

The amplitude of the Gaussian envelope.

→ For an 'arbfreq' pulse, there are three more parameters, detailing the modulation which is to be applied to the pulse. Note that this is the modulation of one arbitrarily chosen frequency by another; these parameters can be switched with the appropriate ones at the beginning of the `Qp` command if desired. The pulse is multiplied by $A \sin(\omega_{ARB} t + \phi)$.

float,

ϕ, the phase of the modulation.

float,

f_{ARB}, the frequency of the modulation. Note that the 1445 will generate the modulation signal

at the frequency (*CWfreq* - f_{ARB}), since the digitally-synthesized signal will be mixed with the CW frequency. Also note that due to the bandpass filters in the analog hardware, f_{ARB} cannot be too far off the Larmor frequency.

float)

A, the amplitude of the modulation.

Qwait(float,

The time to wait, in seconds.

D_i{ 'decouple'},

If this parameter is 'decouple', then the next parameter indicates the number of spins to decouple, and the remaining parameters list the spins to be decoupled. If this parameter is anything else, then the next parameter indicates the number of spins not to decouple, and the remaining parameters list the spins not to be decoupled. If using the option `minimal` in `Qdecouple`, then it is recommended that this parameter be 'decouple', that the next parameter be 0, and that the remaining parameters be omitted.

int,

The number of spins to decouple (or, as noted in the previous paragraph, not to decouple).

int, int . . .)

The spins to be decoupled (or, as noted in the above paragraph, not to be decoupled).

Qread(float,

The time to read, in seconds. This length of time is constrained by the blocksize indicated in `Qdigitize`

D_i{'decouple'},

If this parameter is 'decouple', then the next parameter indicates the number of spins to decouple, and the remaining parameters list the spins to be decoupled. If this parameter is anything else, then the next parameter indicates the number of spins not to decouple, and the remaining parameters list the spins not to be decoupled. If using the option `minimal` in `Qdecouple`, then it is recommended that this parameter be 'decouple', that the next parameter be 0, and that the remaining parameters be omitted.

int,

The number of spins to decouple (or, as noted in the previous paragraph, not to decouple).

int, int, int . . .)

The spins to be decoupled (or, as noted in the above paragraph, not to be decoupled).

Qsynch(D_i{'leftjustified', 'rightjustified', 'centered'})
Qendsynch()

These commands, when placed around lists of `Qx`, `Qy`, and `Qp` pulses that target different spin species, can be used to synchronize the various pulses with respect to one another. The option 'leftjustified' means that all the framed pulses start at the same time (possibly ending at different times). The option 'rightjustified' insures that all the framed pulses end at the same time. The option 'centered' insures that the framed pulses' centers all coincide.

q programs that perform Grover's algorithm

Here are the 27 programs that perform Grover's algorithm on two qubits, recognizing the state $|11\rangle$ (e.g., the fourth state, in the lexicographic ordering). The molecule is $^{13}CHCl_3$, chloroform.

As detailed in the section on temporal averaging, as well as in the simulation section, temporal

averaging requires three different perparations (labeled *a*-*c* below). Also, to retrieve all the information from the density matrix requires nine different readout steps (labeled *A*-*I* below). Hence the 27 programs, which are represented below.

q code for the three different preparations is as follows:
a - No preparation. Thus no code needed.
b - Permute ($01 \rightarrow 10 \rightarrow 11 \rightarrow 01$), see **Appendix B** for justification.

```
% code to accomplish preparation b
Qy(2,90,square)
Qwait(0.002325581395349, dec, 0)
Qy(1,-90,square)
Qx(1,-90,square)
Qy(1,90,square)
Qy(2,-90,square)
Qx(2,-90,square)

Qy(1,90,square)
Qwait(0.002325581395349, dec, 0)
Qy(2,-90,square)
Qx(2,-90,square)
Qy(2,90,square)
Qy(1,-90,square)
Qx(1,-90,square);
```

c - Permute ($01 \rightarrow 11 \rightarrow 10 \rightarrow 01$).

```
% code to accomplish preparation c
Qy(1,90,square)
Qwait(0.002325581395349, dec, 0)
Qy(2, -90, square)
Qx(2, -90, square)
Qy(2, 90, square)
Qy(1,-90, square)
Qx(1,-90, square)

Qy(2,90, square)
Qwait(0.002325581395349, dec, 0)
Qy(1,-90,square)
Qx(1,-90,square)
Qy(1,90,square)
Qy(2,-90,square)
Qx(2,-90,square)
```

q code for the nine different readout steps is given below.
A - No rotation. Hence no code.
B - Rotate 1 by Y(π/2).

```
Qy(1, 90, square)
```

C - Rotate 2 by Y(π/2).

```
Qy(2, 90, square)
```

D - Rotate 1 by X(π/2).

```
etc.
```

E - Rotate 2 by X(π/2).
F - Rotate 1 and 2 each by X(π/2).
G - Rotate 1 and 2 each by Y(π/2).
H - Rotate 1 and 2 by X(π/2) and Y(π/2), respectively.
I - Rotate 1 and 2 by Y(π/2) and X(π/2), respectively.

The 27 programs therefore all look like this:

```
Qname('g2|11>ad')
Qsample('13CHCl3')
Qversion('1.0 esb')
Qdecouplingstyle('minimal')
```

```
Qnumqbs(2)
Qspecies(1,'1H', 26.7510e7, 42574931.880108, 35000000, 1, 0.001, 5)
Qspecies(2,'13C', 6.7263e7, 10705040.871934, 8000000, 0.0159, 0.001, 5)

% insert appropriate preparation – one of a, b, or c

Qx(1,180,square)
Qy(1,90,square)
Qx(2,180,square)
Qy(2,90,square)

% Run the U = DC operation.
Qwait(0.002325581395349, dec, 0)
Qy(2,-90,square)
Qx(2,90,square)
Qy(1,-90,square)
Qx(1,90,square)
Qwait(0.002325581395349, dec, 0)
Qy(2,-90,square)
Qx(2,-90,square)
Qy(1,-90,square)
Qx(1,-90,square)

% insert appropriate readout step – one of A, B, C, D, E, F, G, H, or I
```

The ROS scripting language commands

The ROS scripting language is simple: it merely comprises commands telling ROS how to treat certain lines of *q* code. That's all. There are three types of *q* code: prototype code, sweep code, and search-range code. The first type of code is just copied into the *q* file, and usually includes things like instrument settings, titles and filenames, etc. – although these things can also be searched/swept over, if so desired. Sweep code is used for brute-force searches, and for extracting lots of data. COSY and other multidimensional experiments, for example, are easy to implement using sweeps. Search-range code is for optimizing certain elements, and is useful for shimming and adjusting Larmor frequencies/π-pulse lengths.

Rbegin()
Rend()
Does nothing. Used to frame the ROS script.

Rverbose(on)
If verbose, the optimizing routines display all intermediate results, and other parts of the program are generally more verbose.

Rdelay(int)
Wait a certain time (in milliseconds) after each QOS call, before continuing with the experiment. This lets the spins relax back to thermal equilibrium and lets the electronics settle.

Rqospath(S_x[256])
Location of the QOS program.
Rdatapath(S_x[256])
Where to put any acquired data. (In the examples in **Appendix F**, these paths are explicitly hardcoded in the program code, for clarity.)

Rprotobegin()
Rprotoend()
These commands frame any *q* code which is to be copied in verbatim.

Rrangebegin(S_x[8], float, float)
Rrangeend()

These commands declare a search variable specified by a character string up to 8 characters long, with upper and lower bounds. Any use of the character string will be replaced by the appropriate value of the search variable at each point in the search procedure.

Rsweepbegin(S_x[8], float, float, float)
Rrangeend()

These commands declare a sweep variable specified by a character string up to 8 characters long, with upper and lower bounds. The last parameter to `Rsweepbegin` is the stepsize. Any use of the character string will be replaced by the appropriate value of the sweep variable at each point in the sweep procedure.

Rgoal(D_i{ browse, maxq, extract })
Rmethod(D_i{ easy, simplex, powell, conjgrad, phascorr, cwm })

See **V.1.iii.** for descriptions of these commands.

Rtolerance(float)

For search/optimization ROS scripts, this sets the fractional tolerance for determining convergence.

Appendix E: QOS, algorithms and selected code sequences

QOS flow

QOS loads the indicated *q* file, parses it, and sends various initialization parameters to the 1445, 1437, Hitachi, and shim modules. QOS then calculates the durations and start-times for each operation (synchronizing if necessary), synthesizes the voltage waveforms (normalizing to the output voltage of the 1445), and sends the waveform data to the 1445. Then QOS converts the pulse sequence into state-transition information to be used by the Hitachi microcontroller, which gates the output of the ^{1}H and ^{13}C transmit channels, as well as the pin diode switch state, throughout the NMR experiment. Finally, QOS arms all the modules and starts the computation, reading in the data at the end.

QOS comprises over 5000 lines of code, written in Visual Basic, C, and assembly. It runs on a PC, a Hitachi SH-1 microcontroller, and two HP VXI modules. Therefore although it is customary to present one's code in appendices to one's thesis, this is not done here. The code would take up about 70 pages in 8-point Courier font! However, we give some salient examples of the code in QOS in this section, enough to give a flavor of the program, and indicate some of the structure behind *q*.

Parsing

The heart of QOS is the parser. There exist tools like *flex* and *bison*, which generate code for lexical scanning and parsing respectively, but the *q* language is simple enough that writing the parser in straight C sufficed. Below is a large chunk of the parser code; it gives a flavor of the complexity that QOS acquired as *q* and the Mark I quantum computer matured. Code has been modified to make it more instructive; also, code that is inessential for understanding the structure (or which has some function, but which is peripheral) has been omitted.

The include file, **parse.h**:

```
/* PARSE.H esb */
#ifndef PARSE_INCLUDED
#define PARSE_INCLUDED

/* TYPES */
#define PROTX 0
#define PROTY 1
#define PWAIT 2
#define PREAD 3
#define PSEPA 4
#define PROTP 5
#define TSQUARE 0
#define TEBURP 1
#define THALFGAU 2
#define TGAU 3
#define TDECOUPLE 4
#define TNODECOUPLE 5
#define TARBFREQ 6
#define DMINIMAL 0
#define DMAXIMAL 1
#define DSOFT 2
#define TSOLITUDE 0
#define TLEFTJUSTIFIED 1
#define TCENTERED 2
#define TRIGHTJUSTIFIED 3

typedef struct _Op {
```

```
        struct _Op *prev;
        struct _Op *next;
        int number;
        int operation;
        int type;
        double  fparam[15];
        double time;
        int iparam[MAXNUMQBS+1];
        int typesynch;
        int issegment;
        struct _Segment *segment;
        char command[LINESIZE];
} Op;

typedef struct _Segment {
        struct _Segment *prev;
        struct _Segment *next;
        int number;
        struct _Op *op;
        int length;
        char name[11];
        ViReal64 *volts;
} Segment;

typedef struct _Species {
        struct _Species *prev;
        struct _Species *next;
        int number;
        char name[8];
        double gyromag;
        double Larmor;
        double cw;
        double sensitivity;
        double scale;
        double maxvolt;
        double duration;
        double T1;
        double T2;
} Species;

/* PROCEDURES */
int LoadProgram(FILE *programfile);
int CreateSegmentsFromOps();
int CreateSequenceFromSegmentsAndOps();
int GetLine(FILE *fileptr, char * line, int length);
Species * GetSpecie(Species * thespecies, int number);
double AngMod360(double A);

#endif
```

And for the main parsing program, **`parse.c`**:

```
/* NMRQC parser. */
/* INCLUDES */
#include "essence.h"
#include "qos.h"
#include "drive1437.h"
#include "drive1445.h"
#include "hitachi.h"
#include "parse.h"

/* VARIABLES */
int Qmaxvolt = 5.00;
char Qname[8];
char Qsample[8];
char Qversion[8];
int Qnumqbs = 0;
int Qdecouplingstyle = DMINIMAL;
Species * TheSpecies;
Species * curspecies;
double **QJcouplings;
Op * TheOps;
Op * curop;
Segment * TheSegments;
Segment * cursegment;
```

```
char * segmentstring;
char * dwellstring;
ViReal64 * TheVolts;
ViReal64 * synchelement;
ViReal64 * TheWaitVolts;
int numops = 0;
int numtransitions = 0;
int numsegments = 0;
double curtime = 0;
double framesize = 0, delay1445 = 0, ringdowndelay = 5e-6;
int mod1437 = 1, ext1437=0, mod1445 = 1, modhitachi = 1, modshims = 0;
char message[256];
char Tsquare[] = "square";
char Teburp[] = "eburp";
char Tgaussian[] = "gaussian";
char Thalfgau[]  = "halfgaussian";
char Tdecouple[] = "decouple";
char Tarbfreq[] = "arbfreq";
char comment[]="%";
char Dminimal[] = "minimal";
char Dmaximal[] = "maximal";
char Mcont[] = "continuous";
char Mtrig[] = "triggered";
char Marm[] = "armed";
char Msimul[] = "simulateonly";
char Mverb[] = "verbose";
char Minstant[] = "instantreplay";
char Mnotinstant[] = "notinstantreplay";
char M1437[]="1437";
char Mext1437[] = "ext1437";
char M1445[]="1445";
char Mhitachi[] = "hitachi";
char Mshims[] = "shims";
char Mdecimate[] = "decimate";
char Mspecifytesla[] = "specifytesla";
char Mspecifyduration[]  = "specifyduration";
char MspecifyJtimes[] = "specifyJtimes";
char MspecifyJcouplings[] = "specifyJcouplings";
char Mcomplex[] = "complex";
char Mreal[] = "real";
char Mpowerspectrum[] = "powerspectrum";
char Mtimedomain[] = "timedomain";
char Mfft[] = "fft";
char Mleftjustified[] = "leftjustified";
char Mrightjustified[] = "rightjustified";
char Mcentered[] = "centered";
#define MPOWERSPECTRUM 1
#define MTIMEDOMAIN    2
#define MFFT           3
int dataform;
int iM_;
ViReal32 * thedata;
double readduration;
long totalmemory, readmemory,emptycellmemory,waittime,readmemory_bytes;
ViReal64 centerfreq;
ViInt16 signalbandwidth = 0;
ViInt16 voltrange = 9;
ViInt16 decimateon;
ViInt32 blocksize;
int Mdurationinsteadoftesla = 1;
int MJcouplingsinsteadofJtimes = 0;
ViInt16 dtype;
ViInt16 resolution;
int numshims = 4;
double *shimcurrents;

/* PROCEDURES */
int LoadProgram(FILE *programfile) {
       char line[LINESIZE];
       char wholeline[LINESIZE];
       int curlinenum;
       char delimiters[] =" ,;()[]{}'\t\n\r";
       char stringdelimiters[] = "'\n()";
       char *token;
       int i,j;
```

```
int globalsynchstate=0;
extern int Mreplay;
extern int atten1,atten2;

logprint("Beginning to load program...\n");

logprint("Allocating memory...\n");
NewList(TheOps, Op);
NewList(TheSegments, Segment);
NewList(TheSpecies, Species);
curop = TheOps;
cursegment = TheSegments;
curspecies = TheSpecies;

logprint("Parsing program...\n");
if(Mverbose) {
        logprint("Verbose: printing out program:\n");
        logprint("--------------------------------------------------------------------\n");
}
curlinenum = 0;
while (GetLine(programfile, wholeline, LINESIZE-1)) {
        if(Mverbose) { sprintf(message, "| %s", wholeline); logprint(message); }
        strcpy(line,wholeline);
        curlinenum++;
        if(!(token = strtok(line, delimiters)))
                continue;
        if(!(strcmp(token, "Qbegin")) || !(strcmp(token, "Qend"))) {
                continue;
        } else if (token[0]==comment[0]) {
                continue;
        } else if (!(strcmp(token, "Qname"))) {
                if(!(token = strtok(NULL, stringdelimiters)))
                        logexit(-9);
                strncpy( Qname, token, 8);
                continue;
        } else if (!(strcmp(token, "Qsample"))) {
                if(!(token = strtok(NULL, stringdelimiters)))
                        logexit(-9);
                strncpy( Qsample, token, 8);
                continue;
        } else if (!(strcmp(token, "Qversion"))) {
                if(!(token = strtok(NULL, stringdelimiters)))
                        logexit(-9);
                strncpy( Qversion, token, 8);
                continue;
        } else if (!(strcmp(token, "Qdecouplingstyle"))) {
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                if(strstr(Dminimal, token)==Dminimal) {
                        Qdecouplingstyle = DMINIMAL;
                } else if(strstr(Dmaximal, token)==Dmaximal) {
                        Qdecouplingstyle = DMAXIMAL;
                } else {
                        logprint("This decoupling style is not yet implemented.\n");
                        logexit(-10);
                }
                continue;
        } else if (!(strcmp(token, "Qmode"))) {
                if(!(token=strtok(NULL, delimiters)))
                        logexit(-9);
                if (strstr(Mverb,token)==Mverb) {
                        Mverbose = 1;
                } else {
                        Mverbose = 0;
                }
                if(!(token=strtok(NULL, delimiters)))
                        logexit(-9);
                if (strstr(Msimul,token)==Msimul) {
                        Msimulate = 1;
                } else {
                        Msimulate= 0;
                }
                .
                .
                . [ more mode settings ]
```

```
        .
        .
} else if (!(strcmp(token, "Qnumqbs"))) {
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%d", &Qnumqbs);
        MakeTypeMatrix(QJcouplings, double, Qnumqbs, Qnumqbs);
        continue;
} else if (!(strcmp(token, "QJcouplings"))) {
        for(i=0;i<Qnumqbs;i++)
                for(j=0;j<Qnumqbs;j++) {
                        if(!(token = strtok(NULL, delimiters)))
                                logexit(-9);
                        sscanf( token, "%Lf", &(QJcouplings[i][j]) );
                }
        continue;
} else if (!(strcmp(token, "Qspecies"))) {
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%d", &(curspecies->number));
        if(curspecies->prev && (curspecies->number <= curspecies->prev->number) ) {
                sprintf(message, "Nonmonotonic numbers on line %d.\n", curlinenum);
                logprint(message);
                logexit(-5);
        }
        if(curspecies->number > Qnumqbs) {
                sprintf(message, "Number too large on line %d.\n", curlinenum);
                logprint(message);
                logexit(-7);
        }
        if(!(token = strtok(NULL, stringdelimiters)))
                logexit(-9);
        strncpy(curspecies->name, token, 8);
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%Lf", &(curspecies->gyromag));
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%Lf", &(curspecies->Larmor));
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%Lf", &(curspecies->cw));
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%Lf", &(curspecies->sensitivity));
        if(Mdurationinsteadoftesla) {
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curspecies->duration));
                curspecies->maxvolt = Qmaxvolt;
        } else {
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curspecies->scale));
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curspecies->maxvolt));
        }
        ExtendList(curspecies, Species);
        continue;
} else if (!(strcmp(token, "Qshims"))) {
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%d", &numshims);
        shimcurrents = MakeTypeVector(numshims, double);
        for(i=0;i<numshims;i++) {
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(shimcurrents[i]));
        }
} else if (!(strcmp(token, "Qx"))) {
        numops++;
        curop->number = numops;
        curop->issegment = 1;
        strncpy(curop->command, wholeline, LINESIZE);
```

```
        curop->operation = PROTX;
        curop->typesynch = globalsynchstate;
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%d", &(curop->iparam[0]));
        if(curop->iparam[0] > Qnumqbs) {
                sprintf(message, "Undefined number on line %d.\n", curlinenum);
                logprint(message);
                logexit(-8);
        }
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        sscanf( token, "%Lf", &(curop->fparam[0]));
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
        if(strstr(Tsquare, token)==Tsquare) {
                curop->type = TSQUARE;
        } else if (strstr(Teburp, token)==Teburp) {
                curop->type = TEBURP;
                logprint("Not implemented.\n");
                logexit(-50);
        } else if (strstr(Tgaussian, token)==Tgaussian) {
                curop->type = TGAU;
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[5]));
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[6]));
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[7]));
        } else if (strstr(Thalfgau, token)==Thalfgau) {
                curop->type = THALFGAU;
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[5]));
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[6]));
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[7]));
        } else if (strstr(Tarbfreq,token)==Tarbfreq) {
                curop->type = TARBFREQ;
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[5]));
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[6]));
                if(!(token = strtok(NULL, delimiters)))
                        logexit(-9);
                sscanf( token, "%Lf", &(curop->fparam[7]));
        } else {
                sprintf(message, "Pulsetype unknown on line %d.\n", curlinenum);
                logprint(message);
                curop->type = TSQUARE;
        }
        ExtendList(curop, Op);
        continue;
} else . . .
.
.
. [ similar cases for commands Qy, Qp ]
.
.
} else if(!(strcmp(token, "Qwait"))) {
        numops++;
        curop->number = numops;
        curop->issegment = 1;
        strncpy(curop->command, wholeline, LINESIZE);
        curop->operation = PWAIT;
        if(!(token = strtok(NULL, delimiters)))
                logexit(-9);
```

```
		sscanf( token, "%Lf", &(curop->fparam[0]));
		if(!(token = strtok(NULL, delimiters)))
			logexit(-9);
		if(strstr(Tdecouple, token)==Tdecouple) {
			curop->type = TDECOUPLE;
			if(!(token = strtok(NULL, delimiters)))
				logexit(-9);
			sscanf( token, "%d", &(curop->iparam[0]));
			for(i=1;i<=curop->iparam[0];i++) {
				if(!(token = strtok(NULL, delimiters)))
					logexit(-9);
				sscanf( token, "%d", &(curop->iparam[i]));
			}
		} else {
			curop->type = TNODECOUPLE;
			if(!(token = strtok(NULL, delimiters)))
				logexit(-9);
			sscanf( token, "%d", &(curop->iparam[0]));
			for(i=1;i<=curop->iparam[0];i++) {
				if(!(token = strtok(NULL, delimiters)))
					logexit(-9);
				sscanf( token, "%d", &(curop->iparam[i]));
			}
		}
		ExtendList(curop, Op);
		continue;
	} else if(!(strcmp(token, "Qread"))) {
		numops++;
		curop->number = numops;
		curop->issegment = 0;
		strncpy(curop->command, wholeline, LINESIZE);
		curop->operation = PREAD;
		if(!(token = strtok(NULL, delimiters)))
			logexit(-9);
		sscanf( token, "%Lf", &(curop->fparam[0]));
		if(!(token = strtok(NULL, delimiters)))
			logexit(-9);
		if(strstr(Tdecouple, token)==Tdecouple) {
			curop->type = TDECOUPLE;
			if(!(token = strtok(NULL, delimiters)))
				logexit(-9);
			sscanf( token, "%d", &(curop->iparam[0]));
			for(i=1;i<=curop->iparam[0];i++) {
				if(!(token = strtok(NULL, delimiters)))
					logexit(-9);
				sscanf( token, "%d", &(curop->iparam[i]));
			}
		} else {
			curop->type = TNODECOUPLE;
			if(!(token = strtok(NULL, delimiters)))
				logexit(-9);
			sscanf( token, "%d", &(curop->iparam[0]));
			for(i=1;i<=curop->iparam[0];i++) {
				if(!(token = strtok(NULL, delimiters)))
					logexit(-9);
				sscanf( token, "%d", &(curop->iparam[i]));
			}
		}
		ExtendList(curop, Op);
		continue;
	} else if(!(strcmp(token, "Qsynch"))) {
		if(globalsynchstate) {
			logprint("Can't nest synchronous sequences.\n");
			logexit(-150);
		}
		if(!(token = strtok(NULL, delimiters)))
			logexit(-9);
		if (strstr(Mleftjustified, token)==Mleftjustified) {
			globalsynchstate = TLEFTJUSTIFIED;
		} else if (strstr(Mcentered, token)==Mcentered) {
			globalsynchstate = TCENTERED;
		} else if (strstr(Mrightjustified, token)==Mrightjustified) {
			globalsynchstate = TRIGHTJUSTIFIED;
		} else {
			logprint("I don't understand your synchronicity argument.\n");
```

```
                        logexit(-155);
                }
                continue;
        } else if(!(strcmp(token, "Qendsynch"))) {
                if(!globalsynchstate) {
                        logprint("Not in a synchronous sequence.\n");
                        logexit(-151);
                }
                if( (curop->prev->operation == PROTX) || (curop->prev->operation == PROTY)
                 || (curop->prev->operation == PROTP) ) {
                        curop->prev->typesynch*=-1;
                }
                globalsynchstate=TSOLITUDE;
                continue;
        } else {
                sprintf(message,"Unknown operand on line %d.\n", curlinenum);
                logprint(message);
                logexit(-4);
        }
    }
    if(Mverbose) {
        logprint("------------------------------------------------------------------------\n");
        logprint("Verbose: end of program.\n");
    }
    sprintf(message, "Total %d instructions in program [%s] [%s] [%s].\n", numops,
        Qname, Qsample, Qversion);
    logprint(message);
    return 0;
}

int CreateSegmentsFromOps() {
    Species * curspecie;
    int i,curtick,curindex,newsegment,waitinitialized = 0, prevsegstart=0;
    double duration, freq1445,mu,sigma,startsynch,lengthsynch;
    long numticks;
    char cursegstring[11];
    int transindex,insynch=0;
    Op * opsynchstart;
    int synchstarttick,synchendtick,curstartindex;

    logprint("Scheduling times for the various operations, and adjusting phases...\n");
    curtime = 0;
    if(Mverbose) {
        logprint("------------------------------------------------------------------------\n");
        logprint("Verbose: beginning scheduling.\n");
    }
    totalmemory = 0;
    numtransitions=0;
    for(curop = TheOps; curop->next; curop = curop->next) {
        if(Mverbose) { sprintf(message, "| %s", curop->command); logprint(message); }
        switch(curop->operation) {
            case PROTX:
            case PROTY:
            case PROTP:
                curspecie = GetSpecie(TheSpecies, curop->iparam[0]);
                if(curop->operation != PROTP) {
                    if(AngMod360(curop->fparam[0])>180) {
                        curop->fparam[0] = AngMod360(curop->fparam[0])-180;
                        curop->fparam[2] = (double) PI;
                    } else {
                        curop->fparam[0] = AngMod360(curop->fparam[0]);
                        curop->fparam[2] = 0;
                    }
                    if(Mdurationinsteadoftesla) {
                        duration = curop->fparam[0]*RAD_PER_DEG*(curspecie->
                        duration / (PI/2) );
                    } else {
                        duration = curop->fparam[0]*RAD_PER_DEG/(curspecie->
                        scale*curspecie->maxvolt*curspecie->gyromag);
                    }
                } else {
                    duration = curop->fparam[0] * (1/curspecie->Larmor);
                }
                if( curop->typesynch ) {
                    if(0==insynch) {
```

```
                        opsynchstart = curop;
                        insynch=1;
                        startsynch = curtime;
                        lengthsynch = duration;
                }
                if (1==insynch) {
                        if(duration>lengthsynch)
                                lengthsynch=duration;
                        if( curop->typesynch < 0) {
                                insynch=2;
                                curop=opsynchstart->prev;
                        }
                        break;
                }
                if (2==insynch) {
                        curop->fparam[12] = startsynch;
                        curop->fparam[13] = lengthsynch;
                        switch( (int)fabs(curop->typesynch)) {
                                case TLEFTJUSTIFIED:
                                        curop->fparam[1]=duration;
                                        curop->time = startsynch;
                                        break;
                                case TCENTERED:
                                        curop->fparam[1]=duration;
                                        curop->time = startsynch +
                                        lengthsynch/2-duration/2;
                                        break;
                                case TRIGHTJUSTIFIED:
                                        curop->fparam[1]=duration;
                                        curop->time = startsynch +
                                        lengthsynch-duration;
                                        break;
                                default:
                                        logexit(-600);
                                        break;
                        }
                        HitachiTimes[numtransitions]=
                                HitachiTime(curop->time);
                        HitachiPingates[numtransitions++] =
                                SpeciePingates[curspecie->number-1];
                        HitachiTimes[numtransitions] =
                                HitachiTime(curop->time + curop->fparam[1]);
                        HitachiPingates[numtransitions++] =
                                -SpeciePingates[curspecie->number-1];
                        if(curop->typesynch<0) {
                                insynch=0;
                                curtime += lengthsynch;
                                numticks = (int)(floor(outfreq*lengthsynch));
                                totalmemory += numticks;
                        }
                }
        }
        if (!curop->typesynch) {
                if(insynch) {
                        logprint("Error in synchronization.\n");
                        logexit(-459);
                }
                curop->fparam[1] = duration;
                curop->time = curtime;
                HitachiTimes[numtransitions] = HitachiTime(curop->time);
                HitachiPingates[numtransitions++] =
                        SpeciePingates[curspecie->number-1];
                HitachiTimes[numtransitions] = HitachiTime(curop->time +
                        curop->fparam[1]);
                HitachiPingates[numtransitions++] =
                        -SpeciePingates[curspecie->number-1];
                curtime += duration;
                numticks = (int)(floor(outfreq*duration));
                totalmemory += numticks;
        }
        break;
case PWAIT:
        if(insynch) {
                logprint("Can't have a wait in the middle of a synch.\n");
                logexit(-200);
```

```
                        }
                        if(!waitinitialized) {
                                curop->fparam[1] = duration = curop->fparam[0];
                                curop->time = curtime;
                                curtime += duration;
                                numticks = (int) (floor(outfreq*duration));
                                TheWaitVolts = (ViReal64*)malloc(numticks*
                                        (size_t) sizeof(ViReal64));
                        } else {
                                duration = curop->fparam[0];
                                curop->time = curtime;
                                curtime += duration;
                        }
                        HitachiTimes[numtransitions] = HitachiTime(curop->time);
                        HitachiPingates[numtransitions++] = PINGATE_WAITSTATE;
                        waitinitialized = 1;
                        break;
                case PREAD:
                        if(insynch) {
                                logprint("Can't have a read in the middle of a synch.\n");
                                logexit(-201);
                        }
                        curop->time = curtime;
                        HitachiTimes[numtransitions] = HitachiTime(curop->time);
                        HitachiPingates[numtransitions++] = PINGATE_READSTATE;
                        break;
                default:
                        if(2==insynch) {
                                insynch=0;
                        }
                        if(1==insynch) {
                                insynch=2;
                                curop=opsynchstart->prev;
                        }
                        break;
        }
}
if(Mverbose) {
        logprint("------------------------------------------------------------------------\n");
        logprint("Verbose: scheduling and planning complete.\n");
}

logprint("Creating voltages and digital waveform points...\n");
TheVolts = (ViReal64*)malloc( totalmemory * (size_t) sizeof(ViReal64));
curtick = 0;
curindex= 0;
waitinitialized = 0;
numsegments = 0;
newsegment = 1;
insynch=0;
for(curop=TheOps; curop->next; curop= curop->next) {
        switch(curop->operation) {
                case PROTX:
                case PROTY:
                case PROTP:
                        if(newsegment) {
                                if(curindex>prevsegstart) {
                                        AddSegment(&(TheVolts[prevsegstart]),
                                                curindex-prevsegstart, cursegstring);
                                        prevsegstart = curindex;
                                }
                                numsegments++;
                                curop->issegment = 1;
                                curop->segment = (Segment *)malloc((size_t)sizeof(Segment));
                                if(PROTX==curop->operation) {
                                        sprintf(curop->segment->name, "X%d", curtick);
                                        sprintf(cursegstring, "X%d", curtick);
                                } else if (PROTY==curop->operation) {
                                        sprintf(curop->segment->name, "Y%d", curtick);
                                        sprintf(cursegstring, "Y%d", curtick);
                                } else if (PROTP==curop->operation) {
                                        sprintf(curop->segment->name, "P%d", curtick);
                                        sprintf(cursegstring, "P%d", curtick);
                                }
                        } else {
```

```
                                        curop->issegment = 0;
                                }
                                curspecie = GetSpecie(TheSpecies, curop->iparam[0]);
                                duration = curop->fparam[1];
                                if(PROTP==curop->operation) {
                                        freq1445 = fabs(curop->fparam[9] - curspecie->cw);
                                } else {
                                        freq1445 = fabs(curspecie->Larmor - curspecie->cw);
                                }
                                numticks = (int)(floor(outfreq*duration));
                                if ( curop->typesynch ) {
                                        if(insynch==0) {
                                                synchstarttick = (int)(floor(outfreq*
                                                        curop->fparam[12]));
                                                synchendtick = (int)(floor(outfreq *
                                                        (curop->fparam[12]+curop->fparam[13]))) - 1;
                                                for(i=curindex;i<curindex + (synchendtick-
                                                        synchstarttick);i++) {
                                                        TheVolts[i]=0;
                                                }
                                                insynch = 1;
                                        }
                                        if(insynch==1) {
                                                synchelement = (ViReal64*)malloc( numticks *
                                                        (size_t) sizeof(ViReal64));
                                                if(PROTX==curop->operation) {
                                                        for(i=0;i<numticks;i++){
                                                                synchelement[i] = curspecie->maxvolt*
                                sin(2*pi*freq1445 * (i/outfreq + curop->time) + curop->fparam[2]);
                                                        }
                                                } else if (PROTY==curop->operation) {
                                                        for(i=0;i<numticks;i++){
                                                                synchelement[i] = curspecie->maxvolt*
                                        sin(2*pi*freq1445 * (i/outfreq + curop->time) + PI/2 + curop->fparam[2]);
                                                        }
                                                } else if (PROTP==curop->operation) {
                                                        for(i=0;i<numticks;i++){
                                                                synchelement[i] = curop->fparam[10]*
                                    sin(2*pi*freq1445 * (i/outfreq + curop->time) + RAD_PER_DEG * curop->fparam[8]);
                                                        }
                                                }
                                                switch(curop->type) {
                                                        case TSQUARE:
                                                                break;
                                                        case TGAU:
                                                                mu=numticks/outfreq*curop->fparam[5];
                                                                sigma=numticks/outfreq*
                                                                        curop->fparam[6];
                                                                for(i=0;i<numticks;i++){
                                                                        synchelement[i] *=
curop->fparam[7] * ONEOVERTHESQUAREROOTOFTWOPI *1/sigma exp(-pow(i/outfreq-mu,2)/2/sigma/sigma);
                                                                }
                                                                break;
                                                        case THALFGAU:
                                                                mu=fabs(numticks/outfreq*
                                                                        curop->fparam[5]);
                                                                sigma = numticks/outfreq *
                                                                        curop->fparam[6];
                                                                transindex = (int) (mu*outfreq);
                                                                if(curop->fparam[5]>0) {
                                                                        for(i=0;i<transindex;i++) {
                                                                                synchelement[i] = 0;
                                                                        }
                                                                        for(i=transindex;i<numticks;
                                                                                i++) {
                                                                                synchelement[i] *=
curop->fparam[7]*ONEOVERTHESQUAREROOTOFTWOPI*1/sigma*exp( -pow(i/outfreq-mu, )/2/sigma/sigma);
                                                                        }
                                                                } else {
                                                                        for(i=0;i<transindex;i++) {
                                                                                synchelement[i] *=
curop->fparam[7]*ONEOVERTHESQUAREROOTOFTWOPI*1/sigma*exp( -pow(i/outfreq-mu, 2)/2/sigma/sigma);
                                                                        }
                                                                        for(i=transindex;i<numticks;
                                                                                i++)
```

```
                                                                synchelement[i] = 0;
                                                        }
                                                }
                                                break;
                                        case TARBFREQ:
                                                freq1445 = fabs(curop->fparam[6]-
                                                        curspecie->cw);
                                                for(i=0;i<numticks;i++) {
                                                        synchelement[i] *=
curop->fparam[7]*sin(2*pi*freq1445*(i/outfreq + curop->time) + RAD_PER_DEG*curop->fparam[5]);
                                                }
                                                break;
                                        }
                                        curstartindex = curindex + floor(outfreq *
                                                curop->time) - synchstarttick;
                                        for(i=curstartindex;
                                                i<curindex + floor(outfreq * curop->time) –
                                                        synchstarttick + numticks;
                                                i++) {
                                                TheVolts[i] += synchelement[i-curstartindex];
                                        }
                                        free(synchelement);
                                        if(curop->typesynch <0) {
                                                insynch = 0;
                                                curtime=curop->fparam[12]+curop->fparam[13];
                                                curtick=synchendtick;
                                                curindex += synchendtick - synchstarttick;
                                        }
                                }
                        } else {
                                if(PROTX==curop->operation) {
                        .
                        .
                        .  [ do same stuff for non-synchronized case ]
                        .
                        .
                        break;
                case PWAIT:
                        if(!waitinitialized) {
                                if(curindex>prevsegstart) {
                                        AddSegment(&(TheVolts[prevsegstart]),
                                                curindex-prevsegstart, cursegstring);
                                        prevsegstart = curindex;
                                }
                                numsegments++;
                                curop->issegment = 1;
                                curop->segment = (Segment*)malloc((size_t)sizeof(Segment));
                                sprintf(curop->segment->name, "WAIT");
                                sprintf(cursegstring, "WAIT");
                                duration = curop->fparam[1];
                                emptycellmemory =24;
                                waittime = numticks = (int)(floor(outfreq*duration));
                                emptycellmemory = emptycellmemory - (emptycellmemory % 24);
                                waittime = waittime - (waittime % 24);
                                numticks = numticks - (numticks % 24);
                                for(i=0;i<24;i++){
                                        TheWaitVolts[i] = 0;
                                }
                                AddSegment(TheWaitVolts, 24, cursegstring);
                                waitinitialized =1;
                        } else {
                                if(curindex>prevsegstart) {
                                        AddSegment(&(TheVolts[prevsegstart]),
                                                curindex-prevsegstart, cursegstring);
                                        prevsegstart = curindex;
                                }
                                curop->segment=(Segment*)malloc( (size_t) sizeof(Segment));
                                sprintf(curop->segment->name, "WAIT");
                                sprintf(cursegstring, "WAIT");
                                duration = curop->fparam[1];
                                numticks = (int)(floor(outfreq*duration));
                        }
                        curtick += numticks;
                        newsegment = 1;
                        break;
```

```
                        case PREAD:
                                if(curindex>prevsegstart) {
                                        AddSegment(&(TheVolts[prevsegstart]),
                                                curindex-prevsegstart, cursegstring);
                                        prevsegstart = curindex;
                                }
                                duration = curop->fparam[0];
                                readduration = duration;
                                infreq=(signalbandwidth)?infreq/pow(2,signalbandwidth-1):infreq;
                                numticks = (int)(floor(infreq*duration));
                                readmemory = numticks = pow(2, floor(log(numticks)/log(2))+1);
                                readmemory_bytes = numticks * ((dtype==HPE1437_COMPLEX)?2:1 );
                                if(readmemory_bytes >blocksize) {
                                        logprint("Readmemorybytes is larger than blocksize –
                                                can read in at most one block!\n");
                                        logexit(-1305);
                                }
                                thedata=(ViReal32*)malloc(readmemory_bytes*(sizeof(ViReal32)));
                                break;
                        default:
                                logprint("Invalid segment error.\n");
                                break;
                }
        }
        if(curindex>prevsegstart) {
                AddSegment(&(TheVolts[prevsegstart]), curindex-prevsegstart, cursegstring);
                prevsegstart = curindex;
        }
        logprint("Done creating voltages and digital waveform points...\n");
        sprintf(message, "Allocated %ld memory cells, %ld zeros, total %ld bytes, %ld empty.\n",
                totalmemory,emptycellmemory,totalmemory*sizeof(ViReal64),emptycellmemory*sizeof(Vi
                Real64));
        logprint(message);
        logprint("Done creating the read memory.\n");
        sprintf(message,"Allocated%ld,total%ld bytes.\n",readmemory,readmemory*sizeof(ViReal32));
        logprint(message);
        return 0;
}

int CreateSequenceFromSegmentsAndOps() {
        .
        .
        . [ writes the parsed information into a string for the 1445 – not very instructive ]
        .
        .
}

int GetLine(FILE *fileptr, char * line, int length) {
        int len;
        line[0] = '\0';
        if (fgets(line, length, fileptr)) {
                len = strlen(line);
                if (line[len-1] == '\n')
                        line[len-1] = '\0';
                return 1;
        } else return 0;
}

Species * GetSpecie(Species * thespecies, int number) {
        Species * aspecie;

        for(aspecie=thespecies; aspecie->next; aspecie = aspecie->next) {
                if(aspecie->number== number)
                        break;
        }
        return aspecie;
}

double AngMod360(double A) {
        double a = A;
        while(a>=360)
                a-=360;
        while(a<0.0)
                a+=360;
        return a;
```

```
}
```

Hitachi Microcontroller Code

We used the Hitachi microcontroller in a Low-Cost SH-1 Evaluation Board. It contained an SH7032 Hitachi microcontroller, an EPROM, UARTs/serial ports for communication with a host PC, and an LED. The convoluted code that appears in lines 109-118 of the following C program was designed so that each path through the logic took equal time – an important criterion since we wanted the Hitachi to act as a worthy clock. `SCI_transmit`, `SCI_receive`, and `SCI_init` are serial data transmission protocols due to Rehmi Post. The program flow (as controlled by a small monitor program running on the SH7032) is as follows: `enter -> copy_stuff -> my_main -> exit`.

The cache.c file.

```
/* HITACHI program. Copies itself into the onchip cache
 * for enhanced speed, then executes. E. Boyden. */
/* PORTS * /
#define PADR  (*(volatile short int *)(0x5ffffc0))
#define PBDR  (*(volatile short int *)(0x5ffffc2))
#define PAIOR (*(volatile short int *)(0x5ffffc4))
#define PBIOR (*(volatile short int *)(0x5ffffc6))
#define PACR1 (*(volatile short int *)(0x5ffffc8))
#define PACR2 (*(volatile short int *)(0x5ffffca))
#define PBCR1 (*(volatile short int *)(0x5ffffcc))
#define PBCR2 (*(volatile short int *)(0x5ffffce))
#define delay_cycles(x) ({int __i; for (__i=0; __i < (x) ; __i++);})

#define PIN_GATE    0xE100 /*pin J5-20,19,18,13; port PA15,14,13,8 1110 0001 0000 0000 */
#define ATTEN1_PINS 0x2CDF /*pin J4-14,12,11,8-1;portPB13,11,10,7-0 0010 1100 1101 1111*/
#define ATTEN2_PINSJ4 0x0020 /* pin J4-6; port PB5. 0000 0000 0010 0000 */
#define ATTEN2_PINSJ5 0x1E8F /* J5-17,16,15,14,12,8-5; PA12,11,10,9,7,3,2,1,0 ->
                                0001 1110 1000 1111*/
#define TRIG_1445 0x8000 /* on port B, 15*/
#define TRIG_1437 0x4000 /* PB14 */
#define LED 0x8000        /*PB15 - same as 1445 trigger. */
#define PBCR1_INIT 0x030F /* 0000 0011 0000 1111 */
#define PBCR2_INIT 0x0000 /* 0000 0000 0000 0000  */
#define PACR1_INIT 0x0000 /* 0 */
#define PACR2_INIT 0x3F00 /* 0011 1111 0000 0000 */
#define OUTPUTA_STUFF (PIN_GATE | ATTEN2_PINSJ5 )
#define OUTPUTB_STUFF (ATTEN1_PINS | ATTEN2_PINSJ4 | TRIG_1445 | TRIG_1437 | LED)

#define VECTORLENGTH 200

long      T[VECTORLENGTH];  /* transition times */
short int R[VECTORLENGTH];/* pin-gate values */

#define INST 100000

int my_main() {
        short int i=0,j=0, loop=1, initloop=1, length = 0;
        short int attenuator1,attenuator2,__a,__b;
        char temp[11];

        SCI_init();
        SCIputs("Pulse Programmer 2.0 08/98\n\r");
        PBIOR |= OUTPUTB_STUFF; /* all output */
        PBCR1 &= PBCR1_INIT; /* 0101010101010101 --> turn off reserved/timing output */
        PBCR2 &= PBCR2_INIT; /* 0101010101010101 --> really does nothing */
        PAIOR |= OUTPUTA_STUFF; /* all output */
        PACR1 &= PACR1_INIT;
        PACR2 &= PACR2_INIT;
        PBDR &= ~(TRIG_1445 | TRIG_1437);
       /*Command hierarchy
        commands are:
          - 1 - start pulse sequence
          - 2 - attenuator data for TX board
          - 3 - attenuator data for RX board
          - 4 - pulse program data */
        while(initloop) {
```

```
PBDR &= ~(TRIG_1445 | TRIG_1437);
SCIputs("Command?\n\r");
SCIgets(temp);

i = atoi(temp);
switch(i) {
case 1 :
       SCIputs("Starting step 1...\n\r");
       initloop = 0;
       break;
case 2 :
       SCIputs("Enter bits 1-10.\n\r");
       SCIgets(temp);
       temp[5]='\0'; /* this depends on carriage returns */
       attenuator1 = atoi(temp);
       PBDR &= (~ATTEN1_PINS | attenuator1);
       PBDR |= (ATTEN1_PINS & attenuator1);
       break;
case 3 :
       SCIputs("Enter bits 11-20.\n\r");
       SCIgets(temp);
       temp[5]='\0';
       attenuator2 = atoi(temp);
       PADR &= (~ATTEN2_PINSJ5 | attenuator2);
       PADR |= (ATTEN2_PINSJ5 & attenuator2);
       PBDR &= (~ATTEN2_PINSJ4 | attenuator2);
       PBDR |= (ATTEN2_PINSJ4 & attenuator2);
       break;
case 4 :
       SCIputs("Enter length (up to 400)\n\r");
       SCIgets(temp);
       temp[5]='\0';
       SCIputs("Reading "); SCIputs(temp);
       length = atoi(temp);
       SCIputs("Enter times and values.\n\r");
       for (i=0; i < length; i++) {
              SCIgets(temp);
              temp[10]='\0'; /* carriage returns */
              T[i] = atol(temp);
              SCIgets(temp);
              temp[5]='\0';
              R[i] = atoi(temp);
       }
       SCIputs("Press enter to start.\n\r");
       SCIgets(temp);
       temp[10]='\0';
       PBDR |= TRIG_1445;
       for(__a=0;__a<length;__a++) {
              __b = PADR;
              PADR = (__b | (PIN_GATE & R[__a])) & (~PIN_GATE | R[__a]);
              __b = PADR;
              {
                     int __bb;
                     for(__bb=0;__bb<T[__a];__bb++);
              }
       }
       PBDR |= TRIG_1437;
       break;
case 5:
       PBDR |= TRIG_1445;
       for(__a=0;__a<length;__a++) {
              __b = PADR;
              PADR = (__b | (PIN_GATE & R[__a])) & (~PIN_GATE | R[__a]);
              __b = PADR;
              {
                     int __bb;
                     for(__bb=0;__bb<T[__a];__bb++);
              }
       }
       PBDR |= TRIG_1437;
       delay_cycles(100000); /* one led's worth */
       break;
default:
       SCIputs("[Unrecognized command]\n\r");
       break;
```

```
                }
        }
}

SCIputs (char *s) { while (s && *s) { SCI_transmit (*s++); } }

SCIgets (char *s) {
        char c;
        do { c=SCI_receive();
             *s=c;
             s++; } while ( (c != '\r') && (c != '\n') );
}

int exit() {
        again:
        asm ("sleep");
        goto again;
}
```

The LD file (for linking the program with a consistent memory map), due to Rehmi Post.

```
OUTPUT_FORMAT("symbolsrec")
OUTPUT_ARCH(sh)
INPUT(libm.a libc.a libgcc.a libc.a libgcc.a)

MEMORY
{
  rom   : ORIGIN = 0x00000000, LENGTH = 64k
  ram   : ORIGIN = 0x0A000000, LENGTH = 128k
  cache : ORIGIN = 0x0F000000, LENGTH = 8k
}

SECTIONS
{
  vect_seg :
  {
    *(.vect);
  } > rom

  __vect     = ADDR(vect_seg);
  __vect_end = ADDR(vect_seg) + 1024;

  vrom_end = ((SIZEOF(vect_seg) + 1023) & ~ 1023);

  cache_text_seg : AT(vrom_end)
  {
    cache.o(.text);
  } > cache

  __cache_text  = ABSOLUTE(ADDR(cache_text_seg));
  __cache_text_end = ((ABSOLUTE(ADDR(cache_text_seg)) + SIZEOF(cache_text_seg) + 255) & ~ (255));

  ctext_end = vrom_end + ((SIZEOF(cache_text_seg) + 255) & ~ 255);

  rom_text_seg ctext_end : AT(ctext_end)
  {
    CREATE_OBJECT_SYMBOLS;
    *(.init);
    *(.text);
    *(.strings);
  } > rom

  __rom_text  = ABSOLUTE(ADDR(rom_text_seg));
  __rom_text_end = ((ABSOLUTE(ADDR(rom_text_seg)) + SIZEOF(rom_text_seg) + 255) & ~ (255));

  rtext_end = ctext_end + ((SIZEOF(rom_text_seg) + 255) & ~ 255);

  data_seg : AT(rtext_end)
  {
    *(.data);
  } > cache

  __data     = ABSOLUTE(ADDR(data_seg));
  __data_end = ((ABSOLUTE(ADDR(data_seg)) + SIZEOF(data_seg) + 255) & ~ (255));
```

```
  data_end = rtext_end + ((SIZEOF(data_seg) + 255) & ~ 255);

  uninit_data_seg : AT(data_end)
  {
    *(.bss);
    *(COMMON);
    _end = . ;
  } > cache

  __uninit_data  = ABSOLUTE(ADDR(uninit_data_seg));
  __uninit_data_end = ABSOLUTE(ADDR(uninit_data_seg)) + SIZEOF(uninit_data_seg);

  stack_seg 0x0F001ff4:
  {
    *(.stack);
  } > cache
  __stack = ADDR(stack_seg);
}
```

The init.c file, due to Rehmi Post.

```
/* SETUP PORTS */
#define PADR  (*(volatile short int *)(0x5ffffc0))
#define PBDR  (*(volatile short int *)(0x5ffffc2))
#define PAIOR (*(volatile short int *)(0x5ffffc4))
#define PBIOR (*(volatile short int *)(0x5ffffc6))
#define PACR1 (*(volatile short int *)(0x5ffffc8))
#define PACR2 (*(volatile short int *)(0x5ffffca))
#define PBCR1 (*(volatile short int *)(0x5ffffcc))
#define PBCR2 (*(volatile short int *)(0x5ffffce))
#define delay_cycles(x) ({int __i; for (__i=0; __i < (x) ; __i++);})
#define INST 100000

int enter ()
{
  PBIOR |= 0x8000;
  PBCR1 &= 0x7fff;
  PBDR  |= 0x8000;
  delay_cycles(INST);
  PBIOR |= 0x8000;
  PBCR1 &= 0x7fff;
  PBDR  &= 0x7fff;
  delay_cycles(INST);

  copy_stuff ();
  my_main ();
}

typedef unsigned long ULONG;

copy_stuff ()
{
       ULONG *src;
       ULONG *dst;
       ULONG *end;
       extern ULONG _vect_end, _cache_text, _cache_text_end;

       src = &_vect_end;
       dst = &_cache_text;
       end = &_cache_text_end;
       while (dst < end) {
         *dst++ = *src++;
       }
}
```

Appendix F: ROS, algorithms and selected code sequences

Optimization methods

For the goal `maxQ`, there are three different offered methods, `simplex`, `powell`, and `conjgrad`. Each of these is briefly described below. References for this section are (Ger98) (Pre). Code in the C language is given in (Pre) for all of these procedures.

The Nelder-Mead downhill simplex method, `simplex`

The downhill simplex method minimizes a function $f(\vec{x})$ over a d-dimensional space X which can be described by a simplex – that is, the convex volumetric hull of a complete graph on $d + 1$ points. Simplices in 2 and 3 dimensions correspond to triangles and tetrahedra, respectively. In each step of the Nelder-Mead algorithm, the simplex is reflected, contracted, or expanded, depending on the values that $f(\vec{x})$ takes on the corners; the most common step in the Nelder-Mead algorithm is to take the vertex $\vec{x}$ at which $f(\vec{x})$ is a maximum, and move it to some $\vec{x}\,'$ such that $f(\vec{x}\,') < f(\vec{x})$. The simplex is commonly compared to an amoeba oozing to the minimal point. It is heavy on function evaluations, but is sometimes useful if nothing else seems to work.

Note that the ROS scripting language specifies the search space in terms of upper and lower bounds – that is, a scaled hypercube. Since the `simplex` algorithm requires a search space in the form of a simplicial complex, it automatically picks a simplex containing the search space specified by the ROS script. The wasted volume caused by this automatic-simplex picking is linear in the volume of the specified hypercubic search space.

The Powell Direction Set method, `powell`

Many d-dimensional minimzation algorithms use, as a subroutine, a line-minimization algorithm (which, given a function $f(\vec{x})$, finds the point $\vec{x}_{\min}$ on a line L such that $f(\vec{x}_{\min}) \leq f(\vec{y})$ for all $\vec{y}$ on L). The line-minimization method can be implemented recursively, evaluating $f(\vec{x})$ at several points on a line, then increasing the resolution in the part of the line that seems to be the most promising, and repeating the procedure. Powell's method gives a sequence of vectors in different directions, along which $f(\vec{x})$ is line-minimized. The directions satisfy a *conjugacy requirement*, which makes sure that line-minimizations don't interfere with one another.

The line-minimization routine comprises the `mnbrak`, `brent`, and `linmin` routines from Numerical Recipes.

Conjugate gradient descent, `conjgrad`

Conjugate gradient descent is a variant of the familiar gradient descent algorithm, which finds the minimum of a function $f(\vec{x})$ by moving downhill, e.g. by always following the vector $-\nabla f(\vec{x})$ until reaching a minimum (e.g., a point where the gradient $\nabla f(\vec{x})$ is zero). Conjugate gradient descent tries to prevent successive iterations of the gradient descent algorithm from interfering with one another, by making each iteration move in a different direction from the previous one. In particular, one can try and move down the component of the gradient which is orthogonal to the direction moved in the last iteration. This can prevent, for example, a minimization method from volleying back and forth across a narrow valley.

Since the function we want to minimize, the *Q* function, is not differentiable, we must numerically estimate the derivative. We do this by a variant of a method detailed in (Pre): to find the derivative of a function $f(\vec{x})$ along some coordinate x_i,

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

one can calculate the finite-differences for various values of h, and extrapolate $h \to 0$ using a polynomial model. This is a fairly economical way to compute the gradient of a function, and makes conjugate gradient methods viable.

Comparison

Quite briefly, we can compare the performance of these three optimization algorithms on a four-dimensional space (x, y, z, w) with some dummy functions (note: the GMW electromagnet manual suggests that the inhomogeneity in the magnetic field is fairly smooth, even parabolic). The table lists the number of function evaluations required to find the minimum of each function, with 1% tolerance:

$f(\vec{x}) \to$	$(x + y + z + w)^2$	$(e^x e^y + zw)^2$	$(\sin(x)\cos(y) + z + w)^2$	$(\sin(x)\cos(y) + zw)^2$
`simplex`	77	314	210	210
`powell`	120	154	118	118
`conjgrad`	119	121	123	91

As is clear from the table, evaluating the performance of such a function can be difficult to do analytically. Preliminary trials on the real NMR spectrometer are inconclusive.

The quality (*Q* function)

The *Q* function is described by the code given below (which has been modified for readability). I assume that the time-domain data coming back from the HPE1437 is complex, not real. `mrqmin` and `four1` are Numerical Recipes routines for performing Levenberg-Marquardt function fitting and the complex Fast Fourier Transform, respectively. Code for `QwriteJitRun` is given after the description of `quality`; this mysterious-sounding procedure synthesizes the *q* program to be passed to QOS. (Roughly, the name `QwriteJitRun` is derived from salient letters from the not-very-eloquent phrase '*q* program writeout, with just-in-time compilation and running.') `vector`, `matrix`, `free_vector`, and `free_matrix` are Numerical Recipes primitives for allocating and deallocating memory structures.

```
#define freqout(i) ( (infreq)*( ((float)(i)>(float)nsamp/2) ? \
                        (-1 + (float)(i)/(float)nsamp) :    \
                        (0 + (float)(i)/(float)nsamp) ) )
float quality(float *point) {
       int dtype,i,nsamp,sanityiters;
       float infreq;
       static int counts=0;
       static int calledalready=0;
       static float * thedata;
       static float * thespectrum;
       char outfilename[] = "c:\\users\\nmr\\qos\\data\\grover.dat";
       FILE * datafile;

       float DCoffsetR, DCoffsetI;
```

```
float bw,peakfreq,peakval,sd,background;
int peakindex;
float widthptR,widthptL;
float a,w0,theta,A;

/* Write the q program and run QOS. */
QWriteJitRun(point);

/* Acquire data. */
datafile = fopen(outfilename, "r");
fscanf(datafile, "%d\n", &dtype);
fscanf(datafile, "%f\n", &bw);
infreq = (bw==0) ? 20000000 : 20000000/pow(2,bw-1);
fscanf(datafile, "%d\n", &nsamp);
if(!calledalready) {
        thedata = (float*)malloc( 2* nsamp * (size_t) sizeof(float));
        thespectrum = (float*)malloc(nsamp * (size_t) sizeof(float));
        calledalready = 1;
}
for(i=0;i<2* nsamp;i++) fscanf(datafile, "%f\n", &(thedata[i]));
fclose(datafile);

/* Remove the DC offset. */
DCoffsetR = DCoffsetI = 0;
for(i=0;i<nsamp;i++) {
        DCoffsetR += thedata[2*i];
        DCoffsetI += thedata[2*i+1];
}
DCoffsetR /= (float)nsamp; DCoffsetI /= (float)nsamp;
for(i=0;i<nsamp;i++) {
        thedata[2*i] -= DCoffsetR;
        thedata[2*i+1]-= DCoffsetI;
}

/* Numerical recipes Fast Fourier Transform. */
four1(thedata-1, nsamp, 1);
/* Compute the power-spectral density - naive method */
for(i=0;i<nsamp;i++)
        thespectrum[i] = thedata[2*i]*thedata[2*i] + thedata[2*i+1]*thedata[2*i+1];

/* Perform linear approximation before curve-fitting */
peakfreq = 0;
peakindex = 0;
background = 0;
for(i=0;i<nsamp;i++) {
        if(thespectrum[i]>thespectrum[peakindex]) peakindex = i;
        background += thespectrum[i];
}
background /= (float)nsamp;
peakfreq = freqout(peakindex);
peakval  = thespectrum[peakindex];
sd = sqrt(background);
for(i=peakindex;i<nsamp;i++) {
        if(thespectrum[i] <(peakval-background)/2) {
                widthptR = ( (peakval-background)/2 - thespectrum[i-1]) /
                           (thespectrum[i] - thespectrum[i-1]) + (i-1);
                break;
        }
}
for(i=peakindex;i>=0;i--) {
        if(thespectrum[i]<(peakval-background)/2) {
                widthptL = ( (peakval-background)/2 - thespectrum[i]) /
                           (thespectrum[i+1] - thespectrum[i]) + i;
                break;
        }
}
peakfreq = (freqout(widthptR) + freqout(widthptL) )/2;

/* Since this part uses Numerical Recipes conventions, we block it off */
{
        int count, lo, hi, npts,iparam[5];
        float *x,*y,*sig, *param,**covar,**alpha, chisq,oldchisq,alamda=-1;

        a=fabs( infreq * ( ( (widthptR>(float)nsamp/2) ? (- nsamp + widthptR) :
                           (widthptR) ) - ( (widthptL>(float)nsamp/2) ?
```

```
                                (- nsamp + widthptL) : (widthptL) ) )/2/(float)nsamp );
                w0=sqrt(peakfreq*peakfreq+a*a) * ((peakfreq>0)?1:-1);
                theta=0;
                A=a*a*peakval/PI/PI;

                param = vector(1,4);
                covar = matrix(1,4,1,4);
                alpha = matrix(1,4,1,4);
                param[1] = w0;
                param[2] = a;
                param[3] = theta;
                param[4] = A;
                iparam[1] = iparam[2] = iparam[3] = iparam[4] = 1;

                lo = max(peakindex - (int)(5*a/infreq*(float)nsamp),0);
                hi = min(peakindex + (int)(5*a/infreq*(float)nsamp),(float)nsamp);
                npts = hi-lo+1;
                x = vector(1,npts);
                y = vector(1,npts);
                sig = vector(1,npts);
                for(i=1;i<=npts;i++) {
                        x[i] = freqout(lo+i);
                        y[i] = thespectrum[lo+i];
                        sig[i]=sd;
                }
                alamda=-1;
                sanityiters=0;
                while(sanityiters++<1000) {
                        mrqmin(x,y,sig,npts,param,iparam,4,covar,alpha,&chisq,Qlm,&alamda);
                        if (fabs(chisq-oldchisq) < 1e-6) {
                                if((++count) > 5) break;
                        }
                        else count = 0;
                        oldchisq = chisq;
                }
                alamda = 0;
                mrqmin(x,y,sig,npts,param,iparam,4,covar,alpha,&chisq,Qlm,&alamda);
                w0 = param[1];
                a = param[2];
                theta = param[3];
                A = param[4];

                free_vector(x,1,npts);
                free_vector(y,1,npts);
                free_vector(sig,1,npts);
                free_vector(param,1,4);
                free_matrix(covar,1,4,1,4);
                free_matrix(alpha,1,4,1,4);
        }

        return Q;
}
```

And now for `QWriteJITRun`, which writes out a *q* file corresponding to the current parameters being optimized, then calls QOS to execute the *q* file. It's important to know that `TheSweeps` is an array of `Sweep` structures,

```
        typedef struct _sweep {
                char name[8];  /* Name of the variable to be swept. */
                double begin;  /* Lower bound. */
                double end;    /* Upper bound. */
                double step;   /* If just browsing or extracting, */
                int numsteps;  /*   then there is a stepsize.     */
                double curval; /* Current value */
        } Sweep;


int QWriteJitRun(float *point) {
        int i, curline;
        char wholeline[LINESIZE], line[LINESIZE], newline[LINESIZE];
        char * token,*strincidence, *endlastincidence, *newlinept;
        char qospath[] = "c:\\users\\nmr\\qos\\qos.exe";
        char qfilename[] = "c:\\users\\nmr\\qos\\grover.q";
        char outfilename[] = "c:\\users\\nmr\\qos\\data\\grover.dat";
```

```
        for(i=0;i<dimension;i++)
                TheSweeps[i].curval = point[i];

        /* Open the output q file. */
        if ( (qfile = fopen(qfilename, "w")) == NULL) {
                sprintf(message,"ros error\nThe qfile %s could not be opened.\n", qfilename);
                return -1;
        }
        for(curline = 0; curline<numlines;curline++) {
                strcpy(wholeline, TheProgram[curline]);
                strcpy(line,wholeline);
                token = strtok(line, delimiters);
                if (token[0]=='R') continue; /*Skip all ROS script lines-look only for q lines. */
                for(cursweep=0;cursweep<numsweeps;cursweep++) {
                        while(strincidence = strstr(wholeline, TheSweeps[cursweep].name)) {
                                endlastincidence = wholeline;
                                newlinept = newline;
                                strncpy(newlinept, endlastincidence,
                                        strincidence-endlastincidence);
                                newlinept += strincidence-endlastincidence;
                                sprintf(newlinept,"%Lf", TheSweeps[cursweep].curval);
                                newlinept += strlen(newlinept);

                                endlastincidence = strincidence + strlen(TheSweeps[cursweep].name);
                                strcpy(newlinept, endlastincidence);
                                strcpy(wholeline, newline);
                        }
                }
                fprintf(qfile,"%s\n", wholeline);
        }
        fclose(qfile);

        /* Now call QOS in order to execute the newly */
        printf("Calling qos..\n");
        sprintf(message, "%s %s %s", qospath, qfilename, outfilename);
        Sleep(Rdelay); /* The delay to wate before executing the q program via QOS. */
        system(message);
        return 0;
}
```

Extraction methods

For the goal `extract`, the phase correction algorithm can be used to apply a linear phase correction. We start off by performing the Levenberg-Marquardt fit as shown in the quality (*Q*) function `quality` (in the previous section), but instead of just returning `Q` at the end, we perform a linear-phase fitting in the vicinity of the peak. We use the Numerical Recipes `fit` command, which does a linear fit to a dataset. We replace the `return Q;` command by the following block:

```
        {
                int lo, hi;
                char corrdatafile[]="c:\\users\\nmr\\qos\\data\\phasecorrected.dat";

                peakindex = (w0<0) ? (nsamp*(1 + w0/infreq)) : (nsamp*(w0/infreq));

                thephases = (float*)malloc( nsamp * (size_t)sizeof(float) );
                thedataphasecorr = (float*)malloc( 2 * nsamp * (size_t)sizeof(float) );

                for(i=0;i<nsamp;i++)
                        thephases[i] = atan(thedata[2*i+1]/thedata[2*i]);

                /* If we are removing the phase offset entirely, e.g. making it a sinusoid (if
                   we're just subtracting off a specified phase offset, skip this part), then: */
                if(SREAL==submethod) {
                        lo=hi=peakindex;
                        for(i=peakindex;i>max(peakindex-50,0);i--) {
                                if ( fabs(thephases[i] - thephases[i-1])<2 )
                                        lo = i;
                                else
                                        break;
                        }
```

```
                for(i=peakindex;i<(min(peakindex+50,nsamp-1));i++) {
                        if ( fabs(thephases[i+1] - thephases[i])<2 )
                                hi = i;
                        else
                                break;
                }

                /* Perform a linear fit. This code uses Numerical Recipes conventions and
                   is therefore blocked off. */
                {
                        float *x,*y,*sig,siga,sigb,chi2,q;
                        int ndata;

                        ndata = hi-lo+1;
                        x = vector(1,ndata);
                        y = vector(1,ndata);
                        sig = vector(1,ndata);
                        for(i=1;i<=ndata;i++) {
                                x[i] = freqout(lo+i);
                                y[i] = thephases[lo+i];
                                sig[i] = 0.0314;
                        }
                        fit(x,y,ndata,sig,0,&phi0,&phi1,&siga,&sigb,&chi2,&q);

                        free_vector(x,1,ndata);
                        free_vector(y,1,ndata);
                        free_vector(sig,1,ndata);
                }
        }

        /* Now subtract off the phase that was fitted or specified. */
        for(i=0;i<nsamp;i++)
                thephases[i] -= (phi0 + phi1 * freqout(i));
        /* Resynthesize the phase-corrected dataset. */
        for(i=0;i<nsamp;i++) {
                thedataphasecorr[2*i] = thespectrum[i] * cos(thephases[i]);
                thedataphasecorr[2*i+1] = thespectrum[i] * sin(thephases[i]);
        }
        /* Write out the data. */
        datafile = fopen(corrdatafile, "w");
        for(i=0;i<2*nsamp;i++)
                fprintf(datafile, "%f\n", thedataphasecorr[i]);
        fclose(datafile);

        free(thephases);
        free(thedataphasecorr);
}
```

Appendix G: Auxiliary shimming information

Shim Coils

The shim coils were generated using a custom C program that parametrically creates coils in DXF format (the format associated with the computer-aided design program AutoCAD), given the inner diameter, the outer diameter, the number of turns desired, and the resolution to use in approximating the coil as a sequence of line segments. The DXF coils were then imported into a PCB (printed circuit board) layout program (Protel Advanced PCB) and converted to Gerber format for commercial fabrication using etched copper on FR4 epoxy fiberglass. The copper traces were 200 μm wide, with consecutive turns of the spiral separated by 200 μm as well. (Some early shim coils were cut using a LPKF ProtoMat copper PCB mill, then coated with acrylic to prevent oxidation or other damage of the relatively delicate traces; however, this was a very slow process.)

The DXF-generating code is given below, for reference:

```
/* COILGEN.CPP Cuts coils, parametrically specified. E. Boyden */
#include <math.h>
#include <stdio.h>
#include <iostream.h>

#define PI 3.141592653589793

int i;
FILE *outfile;
float xshift=2, yshift=-2.0, xcenter=0, ycenter=0, x=0,y=0,xold=0,yold=0;
float inradius,outradius,resolution,k,kx,ky,tlo,thi,t,dt=0,tsofar;
int numberofturns,tpercentsofar;
char formatstring[] = "0\nLINE\n8\n0\n10\n%f\n20\n%f\n30\n0.0\n11\n%f\n21\n%f\n31\n0.0\n";
float scale = 254;
int main(int argc, char * argv[]) {
        outfile = fopen("coil.dxf","w");
        fprintf(outfile, "0\nSECTION\n2\nENTITIES\n");
        printf("Enter center coordinates x,y (use 0,0 for ordinary PCB fab, use 5,5 for
        Protel).\n");
        scanf("%f, %f", &xcenter, &ycenter);
        printf("Enter scale factor (use 254 for PCB fab, 1 for Protel).\n");
        scanf("%f", &scale);
        printf("Drawing a circular spiral, please enter data.\n");
        printf("For your information the screen has lower coordinates (-1,-1)\n and upper
        coordinates (1,1).\n");
        printf("\nEnter the inner radius of the spiral.\n");
        scanf("%f", &inradius);
        printf("\nEnter the outer radius of the spiral.\n");
        scanf("%f", &outradius);
        printf("\nEnter the number of turns of the spiral.\n");
        scanf("%d", &numberofturns);
        printf("\nEnter the resolution to use per step.\n");
        scanf("%f", &resolution);
        printf("\nSolving equations...\n");
        k = (outradius - inradius)/(2 * PI * (float)numberofturns);
        kx = scale * k;
        ky = scale * k;
        tlo = inradius/k;
        thi = outradius/k;
        printf("Parametrized as %.4f*t*cos(t), %.4f*t*sin(t) from t=%f to %f.\n",k,k,tlo,thi);
        x = 0; y = 0; yold= 0; xold=0;
        tpercentsofar = 0;
        tsofar = tlo + (tpercentsofar+10)/100*(thi-tlo);
        xold = x = kx*tlo*cos(tlo); yold = y = ky*tlo*sin(tlo);
        fprintf(outfile, formatstring,xcenter + xold, ycenter + yold, xcenter + x, ycenter + y);
        for (t=tlo;t<thi;t+=dt) {
                if(t>tsofar) {
```

```
                        printf("%d%% done.\n", tpercentsofar);
                        tpercentsofar +=10;
                        tsofar += .1*(thi-tlo);
                }
                dt = (float)resolution/((float)(k*t));
                x = kx*t*cos(t);
                y = ky*t*sin(t);
                fprintf(outfile,formatstring,xcenter + xold, ycenter + yold,xcenter+x,ycenter+y);
                xold = x; yold = y;
        }
        printf("Done!\n");
        fprintf(outfile, "0\nENDSEC\n0\nEOF\n");
        fclose(outfile);
        return 0;
}
```

Code for the Version II Shims

The code for the version II shim board, which is shorter and clearer than that for the first-model shim board (not presented here, as explained in **Appendix Z**), follows:

```
/*
 *      final shim
 *      Edward Boyden, e@media.mit.edu, NMR QC
 *      3/24/99
 */
#include <16F84.H>

#define LED       PIN_A2
#define SR_SHCP   PIN_A3
#define DEMUX_INA PIN_A4
#define FROM_PC   PIN_B0
#define TO_PC     PIN_B1
#define SR_SIO    PIN_B2
#define SR_STCP   PIN_B3
#define DEMUX_INH PIN_B4
#define DEMUX_IND PIN_B5
#define DEMUX_INC PIN_B6
#define DEMUX_INB PIN_B7
#define DAC_DATA  PIN_A0
#define DAC_CLK   PIN_A1

#use delay(clock=10000000)
#fuses HS, NOWDT, NOPROTECT
#use RS232(baud=38400,xmit=TO_PC,rcv=FROM_PC, parity=N, bits=8)
#define bit short int

byte boardnum = 1; /* shim board #1 */ /* NOT USED IN THIS REV */

byte relay1=0,relay2=0; /* relay1[i] shim i, relay2[i] that of shim (i+8) */

main() {
        byte id=0; /* which shim to modify? */
        byte relay;
        byte lowDAC=0,highDAC=0; /*lowDAC contains bits 0..7, highDAC 8..15 */
        int i;
        byte picker;
        byte latchmap[16] = {10, 11, 8, 9, 14, 15, 12, 13,
                             0, 2,  1, 3, 4,  5,  6,  7};
        byte latchid;

        /* set initial state */
        output_low(LED);
        output_high(SR_SHCP);
        output_low(DEMUX_INA);
        output_low(TO_PC);
        output_low(SR_SIO);
        output_low(SR_STCP);
        output_high(DEMUX_INH);
        output_low(DEMUX_IND);
        output_low(DEMUX_INC);
```

```
output_low(DEMUX_INB);
output_low(DAC_DATA);
output_low(DAC_CLK);

/* blink light */
output_high(LED);
delay_ms(2000);
output_low(LED);
delay_ms(2000);

while(1) {
  if(kbhit()) {
    id = getc() - '0'; /* character id */
    relay = getc() - '0'; /* character 0 or 1*/
    lowDAC = getc();  /* a straight bitstring */
    highDAC = getc();
    for(i=0;i<2;i++) {
       output_high(LED);
       delay_ms(100);
        output_low(LED);
       delay_ms(100);
    }
    delay_ms(1000);
    /* assemble the relay word */
    if(id>7) { /* use relay2 */
      picker = 1 << (id-8);
      relay2 |= relay * picker;
      relay2 &= 255 - ~relay * picker;
    } else { /* use relay1 */
      picker = 1 << id;
      relay1 |= relay * picker;
      relay1 &= 255 - ~relay * picker;
    }

    /* broadcast the relay word, clocking into SR */
    /* shim n --> SR 15-n, so reverse the word*/
    /* but put Q15 in first, so put shim 0 in first -- it all works */
    for(i=0;i<16;i++) {
      if(i>7) {
       if(bit_test(relay2,i-8))
         output_high(SR_SIO);
       else
         output_low(SR_SIO);
      } else {
       if(bit_test(relay1,i))
         output_high(SR_SIO);
       else
         output_low(SR_SIO);
      }
      /* SHCP falling edge */
      output_low(SR_SHCP);
      delay_ms(1);
      output_high(SR_SHCP);
      delay_ms(1);
    }

  /* do a parallel load */
  output_high(SR_STCP);
  delay_ms(1);
  output_low(SR_STCP);
  delay_ms(1);

  /* next, load the DAC as appropriate*/
  for(i=7;i!=0xFF;i--) {
    output_low(DAC_CLK);  /* down-up clock */
    if(bit_test(highDAC,i))
      output_high(DAC_DATA); /* generate a 1 */
    else
      output_low(DAC_DATA); /* generate a 0 */
    delay_ms(1);            /* clock = 500 ns -- 20 instr! */
    output_high(DAC_CLK);
    delay_ms(1);
  }
  for(i=7;i!=0xFF;i--) {
    output_low(DAC_CLK);  /* down-up clock */
```

```
            if(bit_test(lowDAC,i))
              output_high(DAC_DATA); /* generate a 1 */
            else
              output_low(DAC_DATA); /* generate a 0 */
            delay_ms(1);            /* clock = 500 ns -- 20 instr! */
            output_high(DAC_CLK);
            delay_ms(1);
          }

        /* next, latch it in */
        /* this is more complicated. */
        /* first map shim i --> Sxx */ /* then inhibit lo */
        latchid = latchmap[id];
        /* then Sxx -> ABCD */
        if (latchid & 0x01)
          output_high(DEMUX_INA);
        else
          output_low(DEMUX_INA);
        if (latchid & 0x02)
          output_high(DEMUX_INB);
        else
          output_low(DEMUX_INB);
        if (latchid & 0x04)
          output_high(DEMUX_INC);
        else
          output_low(DEMUX_INC);
        if (latchid & 0x08)
          output_high(DEMUX_IND);
        else
          output_low(DEMUX_IND);

        /* then inhibit lo */
        output_low(DEMUX_INH);
        delay_ms(1);
        output_high(DEMUX_INH);
        delay_ms(1);

        /* blink to signify termination */
        for(i=0;i<2;i++) {
            output_high(LED);
            delay_ms(100);
             output_low(LED);
            delay_ms(100);
        }
        delay_ms(1000);
      }
   }
}
```

Appendix Z: The old days

Strictly speaking, the Mark I quantum computer was not the first attempt at an NMR spectrometer by a member of the Physics and Media Group. The "Smartfish" DSP board was intended to be an earth's-field (or low-field) NMR spectrometer (Chong). This was followed by a handbuilt mass of cables and an old magnet (Fletcher), which used a PIC and a few Mini-Circuits boxes to accomplish mixing and amplification. A permanent magnet was then added to the system to make a single channel ^{1}H spectrometer (Maguire). All of these implementations were used to perform spin-echo experiments, with the output displayed on an oscilloscope.

Digital acquisition was then added using a NI-DAQ board (the PCI-MIO-16E-1, which contained a 12-bit 1.25 MHz ADC on board, as well as two 12-bit DACs), which allowed the first pulse programming and signal processing to be performed (Boyden). Software was written first in LabView…

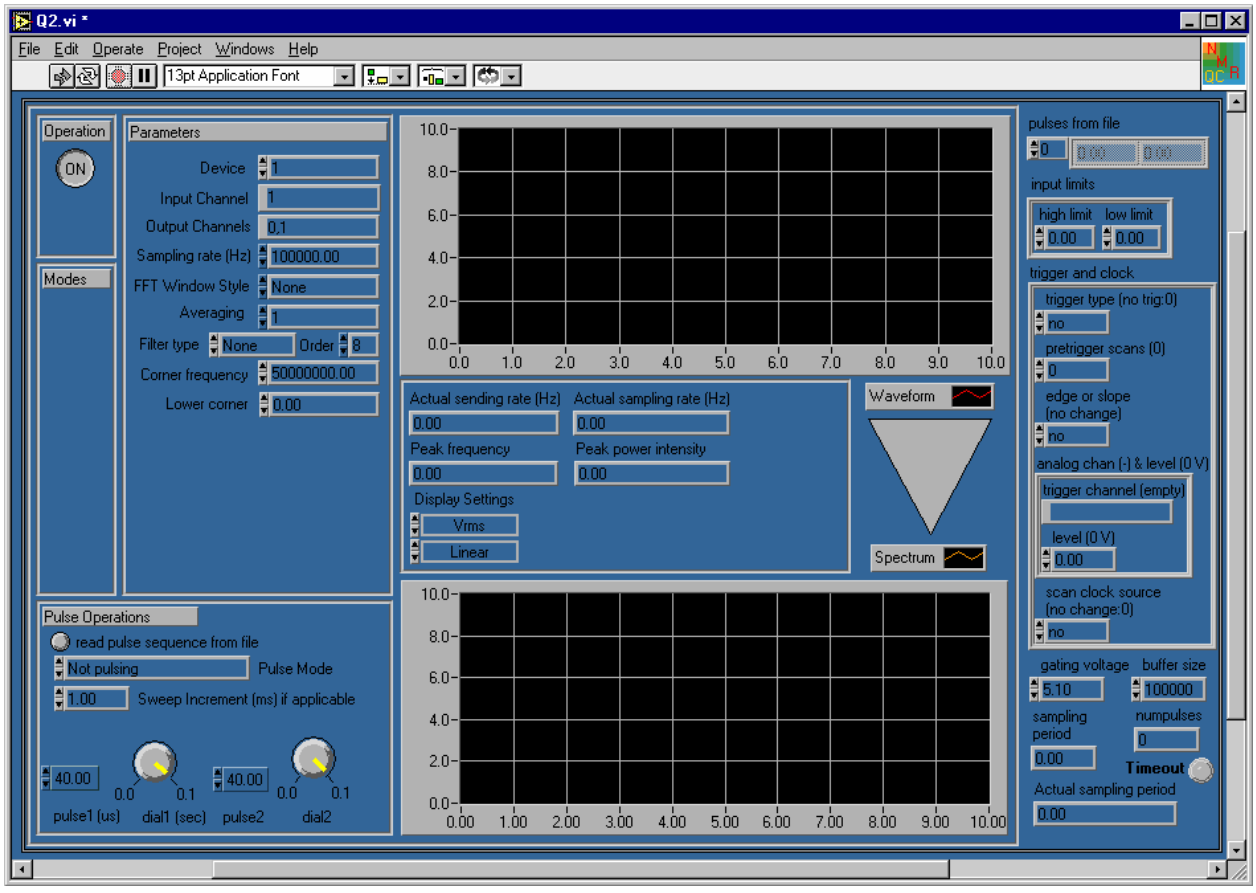


Figure 13: the first interface

…then in C, and a first version of the *q* programming language was written. It was pretty sad: the entire, incomprehensible language comprised the following commands:

```
KEY [all units are volts, microseconds, and Hertz]
# comment
# I is the initialization command
```

```
I outputfreq inputfreq device inchannel outchannelgate outchannel
# L demarcates the beginning of a loop, E marks the end
L loop_id_number numberofrepeats
E loop_id_number
# S is a single pulse, followed by a single wait. No phase can be specified.
S pulselength waitlength
# W specifies a COSY experiment where the pulses are different from π/2 pulses.
W p1 wait1start wait1end wait1increment p2 wait2start wait2end wait2increment
# A specifies an amplitude and a duration. Causes a sinc pulse in frequency space.
A amplitude duration
# T means acquire.
T readtime
# C specifies a COSY experiment.
C wait1start wait1end wait1increment wait2start wait2end wait2increment
```

The parser and pulse generator numbered only a few hundred lines of code, and system integration occurred in a few afternoons. Using this system, the first nontrivial NMR experiment, the 2D HETCOR experiment, was performed in April 1998 (see picture). The program that was run to acquire this data was

```
# the positive-frequency axis COSY program
I 500000.0 500000.0 1 1 0 1
C 100.0 2000.0 50.0 100.0 2000.0 50.0
```

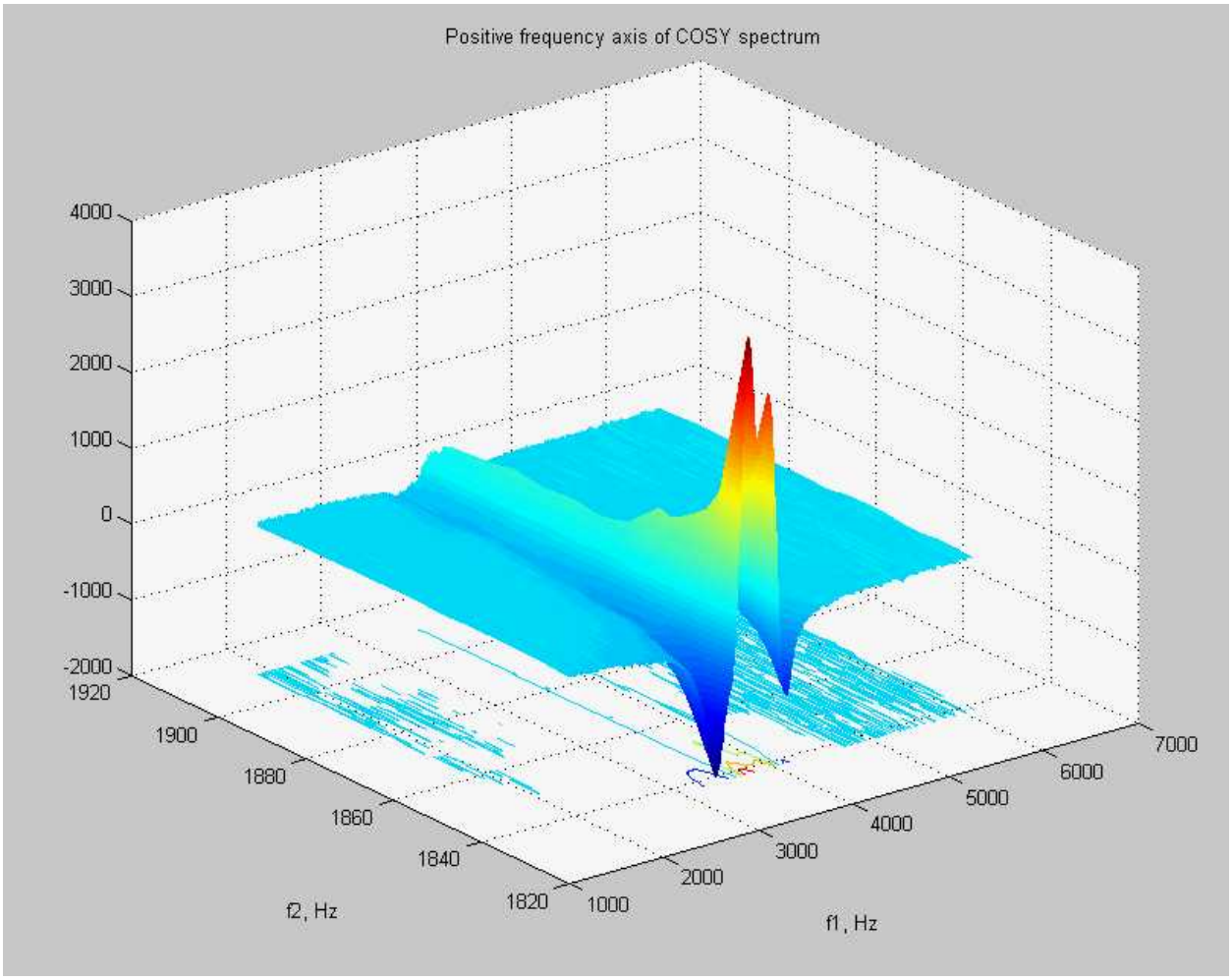


Figure 14: 2D NMR of chloroform

It was not until summer 1998 that printed circuit boards with working NMR circuitry were constructed (Maguire), which became part of the Mark I quantum computer.

The number of false starts throughout this project is interesting, since the assignment all along was to reinvent NMR without looking at past spectrometer designs. As a result lots of little incidents occurred that wasted time but taught much. The experience was analogous to being forced to reinvent the wheel, but with the requirement that the wheel have seven corners, work on ice in zero-gravity conditions, be mass-producible for $100, and have a pretty interface.

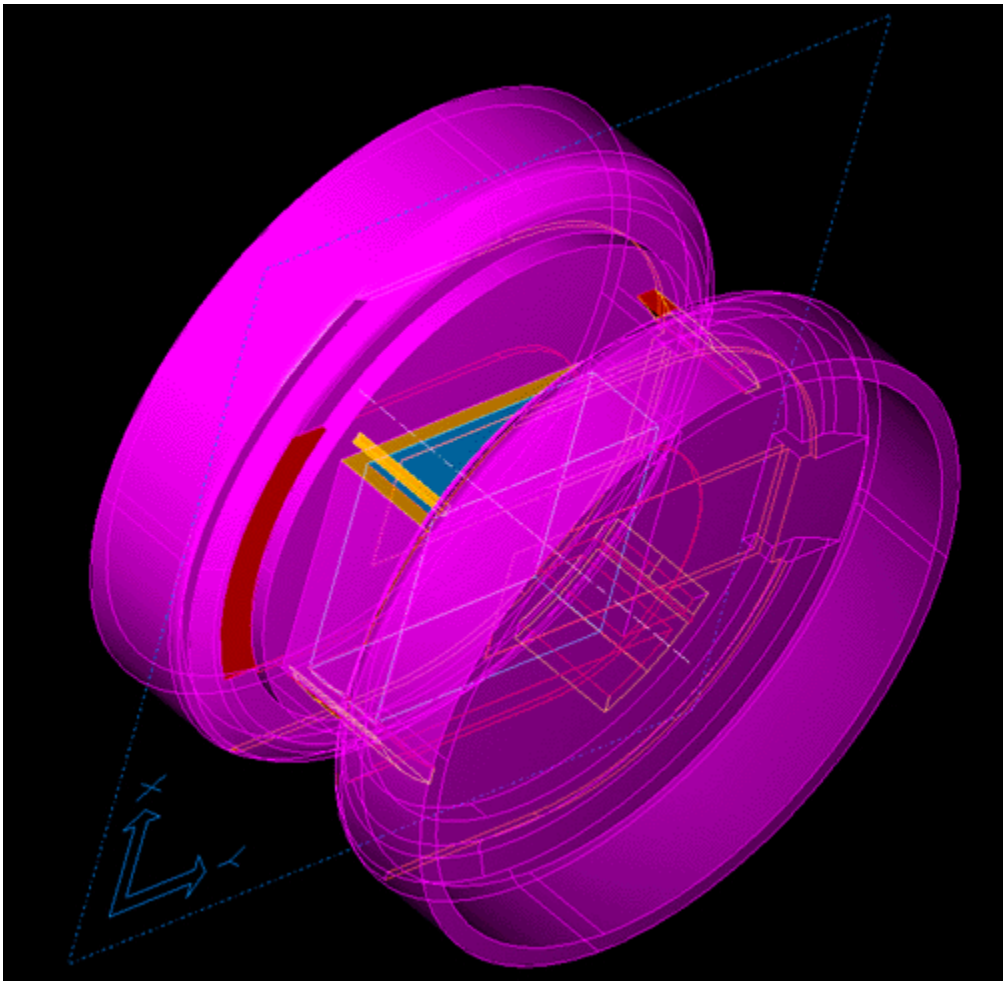


Figure 15: A NMR probe mount designed in SDRC Ideas

For example, I spent a week learning 3D CAD in SDRC-IDEAS in order to make a mount that could hold a set of planar shim coils, as well as a sample and probe-coil (picture shown above). The inside was to be sprayed with a conductive coating (Croshield 4900 Conductive Coating, from Parker Hannifin Corp.), to provide EMI shielding. I printed the mount out in medical-grade ABS plastic on our Stratasys 3D printer; it took over 71 hours to print the 625 layers from the bottom upwards. As it turns out, the tolerances on the 3D printer were so bad that when the mount was placed in the GWM electromagnet, the flexible pole pieces of the electromagnet were forced to tilt slightly, due to the irregularity of the bumpy plastic. As a result of the resultant

magnetic susceptibility perturbations in the vicinity of the sample, the field homogeneity actually became much worse than before! We went back to using the ceramic-spacer module that was provided by the electromagnet manufacturers.

The original shim hardware is pictured below as it appears in PCB layout. It was capable of setting the currents in four coils with values between –20 and 20 mA, with theoretical resolution of about 300 nA.

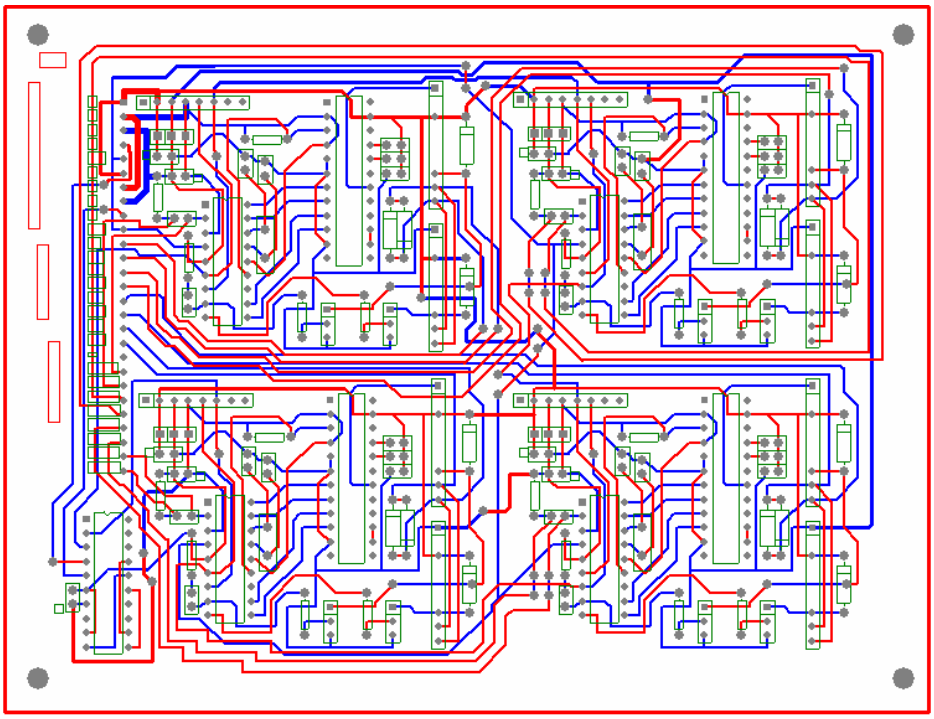


Figure 16: The first shim board

Each of the four copies of the basic shim element contains one PIC16F84 (a small cheap yet surprisingly versatile microcontroller with EEPROM and a small amount of onchip memory), one AD420 current DAC (with output 0 – 20 mA), two relays (for switching the direction of the current in the coil; this assures that the current is not noisy, as would occur with transistor switches), and auxiliary transistors, diodes, power regulators, and LEDs.

In the lower left hand corner is a single MAX233A RS-232 level converter (made by Maxim Integrated Circuits), for enabling communication to a PC. Each PIC passes messages to its neighbor; through this complicated message-passing process, parameters are passed from the PC to the destination PIC, possibly through a number of intermediate PICs. The code is not included, since it is awkward and, although functional, the message-passing routines unnecessarily complicate the design. Eventually the silly message-passing architecture was scrapped, and the shim hardware detailed in **VI.1.ii.** was developed.

References and Bibliography

References like quant-ph refer to the archives at Los Alamos, which is perhaps the greatest single resource on the Internet. See http://xxx.lanl.gov. To quote P. Ginsparg in his UNESCO talk (http://xxx.lanl.gov/blurb/pg96unesco.html),

> The first database, hep-th (for High Energy Physics Theory), was started in August of '91 and was intended for usage by a small subcommunity of less than 200 physicists, then working on a so-called "matrix model" approach to studying string theory and two dimensional gravity…Mermin later described the establishment of these electronic research archives for string theorists as potentially "their greatest contribution to science."…

It is perhaps interesting to think that quantum computing is the first branch of physics which grew up on the Internet. All the papers of note are in quant-ph, and the online community is very vibrant and close-knit.

Quantum Computing

Algorithms (17)

(Abr97) Abrams, D., Lloyd, S., "Simulation of many-body Fermi systems on a universal quantum computer," quant-ph/9703054. 28 Mar 97.

(Ben96b) Bennett, C., Bernstein, E., Brassard, G., Vazirani, U., "Strengths and weaknesses of quantum computing," quant-ph/9701001. 1 Jan 97.

(Bir98) Biron, D., Biham, O., et. al. "Generalized Grover search algorithm for arbitrary initial amplitude distribution." quant-ph/9801066. 11 May 1998.

(Boy96) Boyer, M. "Tight bounds on quantum searching." quant-ph/9605034.

(Cer93) Cerny, V. *Phys. Rev. A.* **48:**116. (1993).

(Cer98) Cerf, N., Grover, L., Williams, C., "Nested quantum search and NP-complete problems." quant-ph/9806078. 23 Jun 1998.

(Far97) Farhi, E., Gutmann, S., "Quantum-mechanical square root speedup in a structured search problem," quant-ph/9711035. 18 Nov 97.

(Gro96) Grover, L., "A fast quantum mechanical algorithm for database search." *Proceedings, STOC 1996, Philadelphia.* 212-219. quant-ph/9605043.

(Gro97b) Grover, L., "Quantum mechanics helps with searching for a needle in a haystack," quant-ph/9706033.

(Gro97) Grover, L., "Quantum computers can search arbitrarily large databases with a single query." quant-ph/9706005.

(Gro97c) Grover, L., "A framework for fast quantum mechanical algorithms." quant-ph/9711043.

(Gro98) Grover, L. "Quantum search on structured problems." quant-ph/9802035.

(Hog97) Hogg, T., "A framework for structured quantum search." quant-ph/9701013. 13 Jan 1997.

(Hog98) Hogg, T., Yanik, M., "Local search methods for quantum computers." quant-ph/9802043.

(Pre96) Beckman, D., Chari, A., Devabhaktuni, S., Preskill, J. “Efficient networks for quantum factoring,” quant-ph/9602016. 21 Feb 96.
(Sho96) Shor, P., “Polynomial-time algorithms for prime factorization and discrete logs on a quantum computer,” quant-ph/9508027. 25 Jan 96. Revised version of the classic 1994 paper.
(Zal98) Zalka, C., “Grover’s quantum searching algorithm is optimal.” quant-ph/9711070.

Coherence (2)

(Chu95) Chuang, I., Laflamme, R., Shor, P., Zurek, W., “Quantum comptuers, factoring, and decoherence.” quant-ph/9503007.
(Unr95) Unruh, W., “Maintaining coherence in quantum computers.” hep-th/9406058.

Computation and quantum complexity theory (3)

(Moo97) Moore, C., Crutchfield, J., “Quantum automata and quantum grammars.” quant-ph/9707031.
(Svo94) Svozil, K. “Quantum computation and complexity theory.” hep-th/9412047 6 Dec 94.
(Svo94b) Svozil, K. “On the computation power of physical systems, undecidability, the consistency of phenomena and the practical uses of paradoxa.” Oct. 25, 1994.

Fault-tolerant quantum computation (5)

(Got97) Gottesman, D., “A theory of fault-tolerant quantum computation.” quant-ph/9702029. v2. 17 Feb 1997.
(Kit97) Kitaev, A., “Fault-tolerant quantum computation by anyons.” quant-ph/9707021.
(Pre97) Preskill, J., “Reliable quantum computers.” quant-ph/9705031.
(Pre97b) Preskill, J., “Fault-tolerant quantum computation.” quant-ph/9712048.
(Sho96b) Shor, P., “Fault-tolerant quantum computation.” quant-ph/9605011.

Gates (10)

(Bar95) Barenco, A., Bennett, C., Cleve, R., DiVincenzo, D., Margolus, N., Shor, P., Sleator, T., Smolin, J., Weinfurter, H., “Elementary gates for quantum computation.” quant-ph/9503016.
(Bar95b) Barenco, A., “A universal two-bit gate for quantum computation.” quant-ph/9505016.
(Bar95c) Barenco, A., “Conditional quantum dynamics and Logic Gates.” quant-ph/9503017.
(Cha95) Chau, H., Wilczek, F., “Simple realization of the Fredkin gate using a series of two-body operators.” quant-ph/9503005. 1 Jun 95.
(Cop94) Coppersmith, D., “An approximate Fourier transform useful in quantum factoring.” *IBM Research Report RC19642*. (1994)
(Deu89) Deutsch, D., “Quantum computational networks.” *Proc. Roy. Soc. Lon. A.* **425**:73. (1989)
(Deu95) Deutsch, D., Barenco, A., Ekert, A., “Universality in Quantum Computation.” quant-ph/9505018. 24 May 1995.
(DiV94) DiVincenzo, D., Smolin, J., in *Proceedings of the Workshop on Physics and Computation.* IEEE Computer Society: Los Alamitos. p. 14. (1994)
(Llo94) Lloyd, S., “Almost any quantum logic gate is universal.” *PRL*. **75**.2:346. 10 July 1995.
(Rec94) Reck, M., Zeilinger, A., Bernstein, H., Bertani, P., “Experimental realization of any discrete unitary operator.” *PRL*. **73:** 58. (1994).

Historical and review (4)

(Bar96) Barenco, A., "Quantum Physics and Computers." quant-ph/9612014. 3 Dec 1996.
(Feyn82) Feynman, R. *Int. J. Theor. Phys.* **21**:467. (1982)
(Llo95) Lloyd, S. "Quantum-mechanical computers." *Scientific American*, **273**:140-145. (1995)
(Ste97) Steane, A., "Quantum computing," quant-ph/9708022. 24 Sep 1997.

Implementation concerns (18)

(Ann) *Annual Reports on NMR Spectroscopy*, Vol. 5A. (1972) Vol. 10B. (1980)
(Cho95) Chong, H. *Towards a Personal Quantum Computer*. M. Eng. Thesis, MIT. (1997)
(Cir95) Cirac, J. and Zoller, P., *Phys. Rev. Lett*. **74**: 4091. (1995)
(Cor98) Cory, D., Mass, W., Price, M., Knill, E., Laflamme, R., Zurek, W., Havel, T., Somaroo, S., "Experimental quantum error correction." quant-ph/9802018. 6 Feb 1998.
(Chu98a) Chuang, I., Gershenfeld, N., Kubinec, M., Leung, D., "Bulk quantum computation with nuclear magnetic resonance: theory and experiment." *Proc. R. Soc. Lond. A* **454**:447-467. (1998)
(Chu98b) Chuang, I., Gershenfeld, N., Kubinec, M., "Experimental Implementation of Fast Quantum Searching," *PRL*. **80**:3408-11. (1998)
(Cor) personal communication.
(Cor97) Cory, D., Fahmy, A., Havel, T., *Proc. Nat. Acad. Sci.* **94**:1634. (1997)
(DiV) personal communication.
(Jon98) Jones, J., Mosca, M., "Implementation of a quantum algorithm to solve Deutsch's problem on a NMR quantum computer." quant-ph/9801027.
(Jon98b) Jones, J., Mosca, M., Hansen, R., "Grover's on homonuclear systems." *Nature*.
(Kan) Kane, B. "An implementation of quantum computation." Lecture at the MIT Laboratory for Computer Science. Physics of Computation series. Fall 1998.
(Kni97) Knill, E., Chuang, I., Laflamme, R., "Effective pure state for bulk quantum computation." quant-ph/9706053. 24 Jun 1997.
(Laf97) Laflamme, R., Knill, E., Zurek, W. H., Catasti, P., S.V.S. Mariappan, S., "NMR GHZ." quant-ph/9709025.
(Llo93) Lloyd, S., "A potentially realizable quantum computer." *Science* **261**:1569-71. 17 Sept 1993.
(Mon95) Monroe, C, Meekhof, D., King, B., Itano, W., Wineland, D., "Demonstration of a universal quantum gate," PRL. **75**:4714. (1995)
(Tuc98) Tucci, R. "A rudimentary quantum compiler." quant-ph/9805015. 7 May 1998.
(Tuc98b) C++ source code for Qubiter. http://www.ar-tiste.com/qubiter.html

NMR and computation (5)

(Ern98) Madi, Z., Bruschweiler, R., Ernst, R., "One- and two- dimensional quantum computing in spin Liouville space." Preprint.
(Fre79) Freeman, R., Kempsell, S., Levitt, M., "Radiofrequency pulse sequences which compensate their own imperfections." *Journal of Magnetic Resonance*. **38**:453-479. (1980)
(Ger97a) Gershenfeld, N., Chuang, L., "Bulk Spin-Resonance Quantum Computation," *Science*. **275**: 350-356. 17 January 1997.

(Sch98) Schulman, L., Vazirani, U., “Scalable NMR Quantum Computation.” quant-ph/9804060. 24 Apr 1998.
(Tyc85) Tycko, R., Pines, A., Guckenheimer, J., “Fixed-point theory of iterative excitation schemes in NMR.” *J. Chem. Phys.* **83**.8. 15 September 1985.

Paradigms (10)

(Amb98) Ambainis, A., Nayak, A., Vazirani, U., “On the space-efficiency of 1-way quantum finite automata.” quant-ph/9804043. 18 Apr 1998.
(Ben97) Benioff, P., “Quantum robots and quantum computers.” quant-ph/9706012. 18 Dec 97.
(Ben97b) Benioff, P., “Models of quantum Turing machines.” quant-ph/9708054. 28 Aug 97.
(Bog97) Boghosian, B., Taylor, W., “Quantum lattice-gas models for the many-body Schrodinger equation.” quant-ph/9701016. v2. 8 Mar 1997.
(Mey96) Meyer, D., “On the absence of homogeneous scalar unitary cellular automata.” quant-ph/960401. v2. 25 Oct 1996.
(Mey96b) Meyer, D., “Quantum mechanics of lattice gas automata I: One-particle plane waves and potentials.” quant-ph/9611005.
(Mey96c) Meyer, D., “Unitarity in one-dimensional nonlinear quantum cellular automata.” quant-ph/9605023.
(Mey96d) Meyer, D., “From quantum cellular automata to quantum lattice gases.” quant-ph/9604003.
(Mey97) Meyer, D., “Quantum mechanics of lattice gas automata II: Boundary conditions and other inhomogeneities.” quant-ph/9712052.
(Nie97) Nielsen, M., Chuang, I. “Programmable quantum gate arrays.” quant-ph/9703032. 18 Mar 97.

Power of (4)

(Ben82) Benioff, P., “QM Hamiltonian models of TMs.” *J. Stat. Phys.* **29**:515 (1982)
(Eke28) Ekert, A., “Quantum computation.” *AIP*. **323**:450.
(Jos97) Josza, R., “Entanglement and quantum computation.” quant-ph/9707034. 17 Jul 97.
(Per97) Peres, A., “Quantum disentanglement and computation,” quant-ph/9707047. 24 Jul 1997.

Quantum dynamics (5)

(Ali96) Alicki, R., et. al. *PRL*. **77**:838. (1996)
(Bal89) Balazs, N., Voros, A., *Ann. Phys.* **190**:1. (1989)
(Bru98) Brun, T., Schack, R., “Realizing the quantum baker’s map on a 3-qubit NMR quantum computer.” quant-ph/9807050.
(Sar90) Saraceno, *Ann. Phys.* **199**:37. (1990)
(Zur94) Zurek, W., Paz, J., *PRL*. **72**:2508. (1994)

Simulation of (7)

(Bmrl) http://bmrl.med.uiuc.edu:8080/software/poma/ Source of the POMA, Product Operators in Mathematica, package. Very useful. For example, here is all the code needed to simulate triple-quantum filtered COSY:

```
spin[1,z]//pulse[90, {0, Pi/3, 2Pi/3, Pi, 4Pi/3, 5Pi/3}]//delay[t1, {{1,
2}, {1, 3}}]//pulse[90, {0, Pi/3, 2Pi/3, 3Pi/3, 4Pi/3, 5Pi/3}]//pulse[90,
{x, x, x, x, x, x}]//receiver[{x,-x,x,-x,x,-x}]//observable//Simplify//sort
```

(Cer97) Cerf, N., Koonin, S., “Monte Carlo Simulation of Quantum Computation.” quant-ph/9703050. 26 Mar 1997.
(Gam) http://gamma.magnet.fsu.edu/ The home page of the GAMMA project, a comprehensive tool for exploring NMR simulation. Includes downloadable source code and binary executables for various platforms.
(Gun) Güntert, P., et al., *Jounal of Magnetic Resonance A*. **101**:103-105. (1993)
(May97) Maymin, P. “The lambda q-calculus can efficiently simulate quantum computers.” quant-ph/9702057. 26 Feb 1997. This is not really a simulation paper, since the lambda q-calculus which Maymin appeals to in order to accomplish his simulation can, for example, solve NP-complete problems in polynomial time, and therefore the paper is misleading.
(Opn) OpenQubit.Org is a group that is making an “Open” quantum computing simulation environment. http://www.openqubit.org
(Smi) Smith, S. *Gamma user’s guide*. http://gamma.magnet.fsu.edu/pdfdox/userman/userman.pdf

Physics and Information

Algorithmic complexity theory (3)

(Cha96) Chaitin, G., *Lecture given at DMTCS'96 at 9 am, Thursday 12 December 1996.* Auckland, New Zealand. http://www.cs.auckland.ac.nz/CDMTCS/chaitin/
(Cha87) Chaitin, G., *Algorithmic Information Theory*. Cambridge University Press. (1987)
(Lloy) Lloyd, S., Zurek, W., “Algorithmic treatment of the spin-echo effect.” Caltech1630.

Coding and communication theory (9)

(Mac77) MacWilliams, F., Sloane, N., *The Theory of Error-correcting Codes*. NorthHolland. (1977)
(Ben96) Bennett, C., DiVincenzo, D., Smolin, J., Wootters, W., “Mixed-state entanglement and quantum error correction.” quant-ph/9604024. (1996)
(Cal96) Calderbank, A., Shor, P., “Good quantum error-correcting codes exist.” quant-ph/9512032. 16 Apr 96.
(Cal97) Calderbank, A., Rains, E., Shor, P., Sloane, N., “Quantum error correction and orthogonal geometry.” quant-ph/9605005.
(Got96) Gottesman, D., “Class of quantum error-correcting codes saturating the quantum Hamming codes.” quant-ph/9604038.
(Kni97) Knill, E., Laflamme, R., “Theory of error-correcting codes.” quant-ph/9604034.
(Sho95) Shor, P., “Scheme for reducing decoherence in quantum memory.” *PR A*. **52**:2493. (1995)
(Ste96) Steane, A., “Error-correcting codes in quantum theory.” *PRL* **77**:793. (1996)
(Ste96b) Steane, A., “Multiple particle interference and quantum error correction.” *Proc. Roy. Soc. London A.*

Complexity of problems (4)

(Che92) Cheeseman, P., Kanefsky, B., Taylor, W., “Computational complexity and phase transitions.” *PhysComp92*, 60-62.
(Hog) Hogg, T., Huberman, B., Williams, C., “Phase Transitions and the Search Problem.”

Artificial Intelligence **81**:1-15. (1996) Appears online at http://www.parc.xerox.com/spl/groups/dynamics/www/specialAIJ/specialAIJ.html
(Hog94) Hogg, T., "Statistical mechanics of combinatorial search." *PhysComp94*. 196-202.
(Kirk) Kirkpatrick, S., Selman, B. "Critical behavior in the satisfiability of random Boolean expressions." *Science*. **254**:1297-1301. (1994)

Computation theory (9)

(Bab98) The Babbage Group, MIT Media Lab, *The Physical Nature of Computing*. Seminar Series. http://physics.www.media.mit.edu/pedagogy/babbage. (1998)
(Cor90) Cormen, T., Leiserson, C., Rivest, R., *Introduction to Algorithms*. MIT Press. (1990)
(Har) Rogers, Hartley. *Theory of Recursive Functions and Effective Computability*.
(Hen) Hennessey, J., Patterson, D., *Computer Architecture: A Quantitative Approach*. Morgan Kaufman. (1996)
(Sac) Sacks, G. *Introduction to logic and recursion theory*. Course 18.511. Spring 1998. Massachusetts Institute of Technology.
(Sip97) Sipser, M., *Introduction to the Theory of Computation*. PWS Publishing:Boston. (1997)
(Sipnote) Amazingly, there is a undecidable problem, *Post's correspondence problem*, which is striking because of its apparent simplicity. Consider a set S of dominos [x | y], where x and y are strings over Σ. The goal is to place the dominos so that the same string reads down both the left and right columns. For example, if [*b* | *ca*], [*a* | *ab*], [*ca* | *a*], [*abc* | *c*] are our four dominos, then we could arrange them as follows:

[*a* | *ab*]
[*b* | *ca*]
[*ca* | *a*]
[*a* | *ab*]
[*abc* | *c*]

and both sides read *abcaaabc*. *It cannot be determined by an algorithm whether an arbitrary finite collection of dominos has a match!* Define PCP = { <*P*> | *P* is a set of dominos which has a match }. Then PCP is undecidable. The proof is by reduction from UTM via accepting computation histories: we can map <*M*, *w*> to a collection of dominos *P* such that a match corresponds to an accepting computation history; the tricky proof involves adding dominos carefully to *P* so that trying to complete the PCP match forces a simulation of *M* on *w*. (PCP is obviously decidable over $\{1\}^*$, but it's not decidable over $\{0, 1\}^*$ -- this ties together several themes that we mentioned earlier.) PCP can be reduced to { <*G*> | *G* is an ambiguous CFG }, so testing whether a CFG is ambiguous is undecidable!

(Sipnote2) Let us define a *regular expression* to be defined by recursive definition under $\cup$ union, $\circ$ concatenation, and * arbitrary repetition, with the atomic elements defined to be the empty language ϕ, the empty string ε, and the elements of an alphabet Σ. For our purposes we adjoin the exponentiation operation ^, so that *R*^3 is $R \circ R \circ R$, the threefold concatenation of *R* with itself: if *R* = 'ab,' then *R*^3 = 'ababab.' It can then be shown that the Turing machine which takes two such regular expressions and tries to compare them, is EXPSPACE-complete (the proof that any EXPSPACE Turing machine can be reduced to this one proceeds via reduction using computation histories: given <*M*,*w*>, simply let *Q* generate all valid substrings of computation histories and *R* generate all strings that are

not computation histories terminating in rejections; if *M* accepts *w* then *R* and *Q* are the same: another diagonal argument!).
(Hop79) Hopcroft, J., Ullman, J., *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley. (1979)
(vL90) van Leeuwen, J., *Handbook of Theoretical Computer Science A: Algorithms and Complexity*. Elsevier. (1990)
(Edw87) Edward, G., Barton, R., Berwick, R., Ristad, E., *Computational Complexity and Natural Language*. Bradford. (1987)

Computation and 'physical' systems (8)

(Arn68) Arnold, V., Avez, A., *Ergodic problems of classical mechanics*. NY: Benjamin (1968).
(Fre82) Fredkin, E., Toffoli, T., *Int. J. Theor. Phys*. **21**:219-253. (1982) (the billiard ball computer)
(Gac83) Gacs, P., "Reliable computation with cellular automata." *J. Comp. Sys. Sci.* **32**:15. (1986)
(Min67) Minsky, M., *Computation: finite and infinite machines*. Prentice-Hall:Englewood Cliffs. (1967)
(Moo90) Moore, C., "Unpredictability and Undecidability in Dynamical Systems." *PRL*. **64**.20:2354-2357. (3 DOF Turing machine)
(Neu52) von Neumann, J., "Probablisitic logics and synthesis of reliable organisms from unreliable components." *Automata Studies*, ed. Shannon, C. Princeton UP. (1952)
(Omo84) Omohundro, S., *Physica* **10D**:128-134. Amsterdam. (1984)
(Wol94) Wolfram, S., *Cellular Automata and Complexity*. Perseus Press. (1994)

Cryptography (2)

(RSA78) Rivest, R., Shamir, A., Adleman, L., "A method for obtaining digital signatures and public key cryptosystems." *Communications of the ACM 21*, **2**:120-126. (1978)
(Lo99) Lo, H., Chau, H., "Unconditional Security of Quantum Key Distribution over Arbitrarily Long Distances." *Science*. 2050-2056. Mar 26 1999.

Entanglement (4)

(Ben96c) Bennett, C., Bernstein, H., Popescu, S., Schumacher, B., *Phys. Rev. A.* **53**:2046. (1996)
(Ben96d) Bennett, C., Brassard, G., Popescu, S., Schumacher, B., Smolin, J., Wootters, W., *PRL*. **76**:722. (1996)
(Pop97) Popescu, S., Rohrlich, D., "Thermodynamics and the measure of entanglement." quant-ph/9610044. v2. 11 Sep 97.
(Unr) Unruh, W., "Is quantum mechanics non-local?" quant-ph/9710032. 10 Oct 97.

Information theory (5)

(Cov91) Cover, T., Thomas, J., *Elements of Information Theory*. J. Wiley and Sons. (1991)
(Dar70) Daroczy, Z. "Generalized information functions." *Information and Control*. **16**:36-51. (1970)
(Sha48) Shannon, C. "A mathematical theory of communication." *Bell Sys. Tech. Journal*. **27**:379-423, 623-656. (1948)
(Sha49) Shannon, C. "Communication in the presence of noise." *Proc. IRE*. **37**:10-21. (1949)

(Wie) Wiener, N. *Cybernetics or Control and Communication in the Animal*. 2nd Ed. MIT Press. (1965)

Information-theoretic physics (3)

(Jay57) Jaynes, E., *Phys. Rev.* **106**:620. (1957) **108**:171 (1957)
(Zur89) Zurek, W., ed. *Complexity, Entropy, and the Physics of Information*. Vol. 8, Santa Fe Institute. (1989)

Mathematics (10)

(Art) Artin, M., *Algebra.* Prentice Hall. (1991)
(Coh89) Cohen, D., *An Introduction to Hilbert Space and Quantum Logic*. Springer Verlag. (1989)
(Dra67) Drake, A., *Fundamentals of Applied Probability Theory*. McGraw Hill. (1967)
(Jef) Jeffreys, H. *Proc. Roy. Soc. London A.* **186**:453-461. (1946) See also Cramer, H., *Mathematical Methods of Statistics*. Princeton UP (1946).
(Mur) Murray, M., Rice, J., *Differential Geometry and Statistics*. Chapman and Hall. (1993)
(Opp) Oppenheim, A., Verghese, G., *6.011 Communications and Control Systems*. Lecture notes. Fall 1997. Massachusetts Institute of Technology.
(Rot97) Rota, G.-C. *Indiscrete Thoughts*. Birkhauser. (1997)
(Sat) Sattinger, D., Weaver, O., *Lie Groups and Algebras with Applications to Physics, Geometry, and Mechanics*. Springer-Verlag. (1986)
(Str) Strang, G., *Introduction to Applied Mathematics*. Wellesley Cambridge (1986)

Nanotech (1)

(Fey61) Feynman, R., "There is plenty of room at the bottom," *Microminiaturization*, ed. Gilbert, H. Reinhold. (1961)

NMR (13)

(Blu85) Blumich, B., Spiess, H., "Quaternions as a practical tool for the evaluation of composite rotations." *J. Magnetic Resonance*. **61**:356-362. (1985)
(BMRL96) NMR Periodic Table. http://bmrl.med.uiuc.edu:8080/MRITable/C.html. (1996)
(Ern90) Ernst, R., Bodenhausen, G., Wokaun, A., *Principles of Nuclear Magnetic Resonance in One and Two Dimensions*. Oxford Univ. Press. (1990)
(Fan) Fan, S., Choy, W., Lam, S., Au-Yeung, S., Tsang, L., Cockram, C. *Analytical Chemistry* **64**:2270-2574. (1992)
(Fre81) Freeman, R., Frenkiel, T., Levitt, M., *J Magn Res*. **44**:409. (1981)
(Fre97) Freeman, R., *Spin Choreography*. Oxford. (1997)
(Ger) Gershenfeld, N. Personal communication.
(Gru) Gruetter, R., et. al., *Magnetic Resonance in Medicine*. **27**:183-188. (1992)
(Kno) This problem, and variants thereof, is colloquially called the 'knobs problem' in the Physics and Media Group. It is a very general engineering problem and goes basically like this: put some knobs on a physical system so that you can manipulate it and interrogate it, until it does something interesting. Then automate it completely, so that all the knobs go away. Now you have something useful.

(Nic) Nicholson, J., Foxall, P., Spraul, M., Farrant, R., Lindon, J. *Analytical Chemistry*. **67**:793-811. (1995)
(Pin96) Navon, G., Song, Y., Room, T., Appelt, S., Taylor, R., Pines, A. "Enhancement of Solution NMR and MRI with Laser-Polarized Xenon." *Science*. **271**:848-1851. March 29 1996.
(Sli90) Slichter, C., *Principles of Magnetic Resonance*. Springer-Verlag. (1990) This book is one of the greatest books for understanding quantum mechanics, perhaps along with Meystre and Sargent, Ashcroft and Mermin, Kittel, and a handful of other tomes that deal with the complexity of real quantum systems while still getting the big picture. Slichter *explains* NMR like physicists used to, with real electrons and real dipoles and real energy. With Slichter, you feel that you are One with the Spin.
(Var) Varian Corporation, *Unity INOVA System Schematics*. Varian Nuclear Magnetic Resonance Instruments. (1998)
(War98) Warren, W., Ahn, S., Mescher, M., Garwood, M., Ugurbil, K., Richter, W., Rizi, R., Hopkins, J., Leigh, J., *Science*. **281**:247-251. 10 July 1998.

Philosophy (1)

(Cha97) Chalmers, D., *The Conscious Mind*. Oxford UP. (1997)

Physics (3)

(Eve73) Everett, H., "The Theory of the Universal Wave Function." in *The Many-Worlds Hypothesis of Quantum Mechanics*, ed. DeWitt, B., Graham, N., Princeton UP. (1973)
(Jack) Jackson, J., *Classical Electrodynamics*, 3rd ed.
(Ger98) Gershenfeld, N., *The Nature of Mathematical Modeling*. Cambridge UP. (1998)

Quantum information theory (15)

(Ben95a) Bennett, C., Brassard, G., Popescu, S., Schumacher, B., Smolin, J., Wootters, W., "Purification of Noisy and Entanglement and Faithful Teleportation via Noisy Channels." quant-ph/9511027. v2. 22 Nov 1995.
(Ben95b) Bennett, C., Bernstein, H., Popescu, S., Schumacher, B., "Concentrating Partial Entanglement by Local Operations." quant-ph/9511030. 21 Nov. 1995.
(Cer97a) Cerf, N., Adami, C., *Phys. Rev. A.* **55**:3371. (1997)
(Cer97b) Cerf, N., Adami, C., "Negative entropy and information in quantum mechanics." *PRL* **79**.26:5194-97.
(Cle96) Cleve, R., DiVincenzo, D. "Schumacher's quantum data compression as a quantum computation." quant-ph/9603009. 7 Mar 96.
(DiV97) DiVincenzo, D., Shor, P., Smolin, J., "Quantum channel capacity of very noisy channels." quant-ph/9706061. 29 Sep 1997.
(DiV97b) DiVincenzo, D., Loss, D., "Quantum information is physical." cond-mat/9710259. 23 Oct 1997. The story of error-correction. In which Landauer, being wrong for the second time, opens up yet new worlds in the theory of physics and computation.
(Jos93) Josza, R., Schumacher, B., "A New Proof of the Quantum Noiseless Coding Theorem."
(Kho73) Kholevo, A., "Bounds for the quantity of information transmitted by a quantum information channel," *Problems of Information Transfer[Transmission]*, IEEE. **9**:3-11, 177. (1973)

(Maa88) Maasen, H., Uffink, J., "Generalized Entropic Uncertainty Relations," *PRL*. **60**:1103-1106. (1988)
(Sch90) Schumacher, B., "Information from quantum measurements", from (Zur89).
(Sch90b) Schumacher, B., *Communication, Correlation, and Complemetarity*. PhD thesis. UT Austin. (1990)
(Sch95) Schumacher, B. "Quantum coding." *Phys. Rev. A.* **51**:2738. (1995)
(Sch95note) Proof by construction: suppose the projection is onto a subspace spanned by eigenstates $|1\rangle,\ldots,|d\rangle$ of ρ, with eigenvalues adding up to > 1-η; these eigenstates are mapped onto the *C* channels and the rest are embedded in the *D* channels, via a unitary transposition operator. Upon traversing the channel, *D* is replaced by $D' \equiv |0\rangle$; tracing the progress of an arbitrary initial message state written in terms of these eigenstates gives the desired result. (There is a lot of algebra involved, as well as the choice of an explicit unitary transposition operator, since this is a constructive proof.)
(Sch96) Schumacher, B., Neilsen, M., "Quantum data processing and error correction." quant-ph/9604022. 22 Apr 96.
(Woo) Wootters, W. *Phys. Rev. D.* **23**:357-362. (1981) Wootters' PhD thesis also has some bearing on this topic.

Quantum logic (2)

(Gri92) Grib, A., Zapatrin, R., "Macroscopic realizations of quantum logics." **Inter. J. Theor. Phys. 31**:1669-1687. (1992) See also Automata simulating quantum logics. **29**:113. (1990)

Quantum mechanics (20)

(Bae) Baez, J. This week in mathematical physics. http://math.ucr.edu/home/baez/#TWF One can spends dozens of hours reading the hundreds of weeks in this diary of one physicist's explorations in radical and incredible theory. Baez is also one of the dominant forces in the popularization of correct physics on the Internet.
(Dir58) Dirac, P., *The Principles of Quantum Mechanics*. Oxford. (1958)
(Fey65) Feynman, R., Hibbs, A., *Quantum mechanics and path integrals*. NY: McGraw-Hill. (1965)
(Gil) Gill, R., Keane, M., "A geometric proof of the Kochen-Specker no-go theorem." *J. Phys. A: Math. Gen.* **29**:289-291. 21 June 1996. The Kochen-Specker proof involves 117 vectors in C^3; since then the number of vectors needed in the proof has been reduced to 31. The Gill-Keane proof reduces the arguments to almost triviality. According to Rota (Rot97), that's when you know that mathematics won't bother that problem anymore.
(Giu) Giulani, D., Joos, E., Kiefer, C., Kupsch, J., Stamatescu, I., Zeh, H., *Decoherence and the Apparance of a Classical World in Quantum Theory*.
(Fuc96) Fuchs, C., *Distinguishability and Accessible Information in Quantum Theory*. PhD thesis. U. New Mexico. quant-ph/9601020. 23 Jan 1996.
(Lib) Liboff, *Introductory Quantum Mechanics.*
(Llo96) Lloyd, S., "A quantum-mechanical Maxwell's demon." quant-ph/9612034. 11 Dec 96.
(Lou73) Louisell, W., *Quantum Statistical Properties of Radiation*. Wiley Classics. (1973)
(Neu55) von Neumann, J. *Mathematical Foundations of Quantum Mechanics*. Princeton UP. (1955) The first few chapters are masterpieces of functional analysis, and the last two

chapters are among the world's clearest explanations of quantum-mechanical entropy, of measurement, and of what quantum mechanics really means.
(Per) Peres, A., *Quantum Theory: Concepts and Methods.*
(Pes) Peskin, M., Schroeder, D., *Quantum Field Theory*.
(Rei65) Reif, F. *Fundamentals of Statistical and Thermal Physics*. McGraw Hill. (1965)
(Sak) Sakurai, J., *Modern Quantum Mechanics*.
(Uhl) Uhlmann, *Rept. on Mathematical Physics*. **9**:273-279. (1976)
(Wei89a,b) Weinberg, S., *PRL* **62**:485. (1989) See also *Ann. of Phys.* **194**:336. (1989)
(Weyl50) Weyl, H., *The Theory of Groups and Quantum Mechanics*. NY: Dover. (1950)
(Woo82) Wootters, W., Zurek, W., *Nature* **299**:802. (1982) (Note that teleportation and transposition do not violate the no-cloning rule.)
(Yur84) Yurke, B., Denker, J., *Quantum network theory*. *PR A* **29**.3:1419-1437. March 1984.

Reversible computation (6)

(Ben73) Bennett, C., "Logical reversibility of computation." *IBM J. Res. Dev.* **17**:525. (1973)
(Ben82) Bennett, C., "The Thermodynamics of Computation: A Review," *Int'l. J. Theor. Phys.* **21**.12:905-940. (1982)
(Ben89) Bennett, C., "Time/space trade-offs for reversible computation." *SIAM J. Comput.* **18**:766. (1989)
(Fre82) Fredkin, E., Toffoli, T., "Conservative logic." *Int'l. J. The. Phys.* **21**:219. (1982)
(Lan61) Landauer, R., "Irreversibility and heat generation in the computing process." *IBM J. Res. Dev.* **5**:183. (1961)
(Per85) Peres, A., "Reversible logic and quantum computers." *PR A.* **32**:3266. (1985)

Speculation on the Physics of Computation (6)

(Abr98) Abrams, D., Lloyd, S., "Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems." quant-ph/9801041. 21 Jan 1998.
(Free) It is known that computing certain invariants for knots, such as the HOMFLY and Jones polynomials, is NP-complete. Therefore if there were a way to observe these things in nature (as suggested by Witten in one of his many theories), NP-completeness would be reduced to triviality. Michael Freedman, at Microsoft Research, is pursuing such ideas. Freedman, who won the Fields medal for showing that the Poincare conjecture was true in 4 dimensions, has used many tools from low-dimensional topology (especially the relations between knot theory and 3- and 4-dimensional space) to discover the properties of such spaces. See Freedman, M., "The topology of four dimensial manifolds." *J Diff Geom*. **17**:357. (1982). Witten, E. "Quantum field theory and the Jones polynomial." *Comm Math P*, **121**:351. (1989).
(Hof) Hofstadter, D., *Godel, Escher, Bach: an Eternal Golden Braid.*
(Jac) Jacobson, J., personal communication.
(Tof) Toffoli, T., *What are nature's natural ways of computing?* (LCS lecture)

Teleportation (2)

(Bra96) Brassard, G., "Teleportation as a quantum computation." quant-ph/9605035. 23 May 1996.
(Dav94) Davidovich, L., et. al., *Phys. Rev. A* **50**, R895 (1994).

Engineering

Analog and NMR design (16)

(Ana) Analog Devices, part datasheets. http://www.analog.com

(Che89) Chen, C., Hoult, D., *Biomedical Magnetic Resonancs Technology*. *Institute of Physics Publishing*. (1989)

(Chi) Chilvers, P., Morris, G. "Automated shimming with normal spectrometer hardware: 3D Profile Edge Shimming." *Journal of Magnetic Resonance* **133**:210-215. (1998)

(Chm90) Chmuray, G., Hoult, D., "The Ancient and Honourable Art of Shimming." *Conc. in Mag. Res.* **2**:131-149. (1990)

(Fuk81) Fukushima, E., Roeder, S., *Experimental Pulse NMR.* Addison-Wesley. (1981)

(Far71) Farrar, T. and Becker, E., *Pulse and Fourier Transform NMR.* Academic Press. (1971)

(Gre98) Greenberg, Y. "Application of superconducting quantum interference devices to nuclear magnetic resonance." *Reviews of Modern Physics*. **70**:1. January 1998.

(Hay96) Hayward, W., *Introduction to Radio Frequency Design*. American Radio Relay League. (1996)

(Hou76) Hoult, R. and Richards, R., "The signal to noise ratio of the nuclear magnetic resonance experiment." *J. Magn. Reson*. **24**:71-85.

(HPRF) Hewlett-Packard RF and Microwave Application Notes. One of the most useful resources assembled on the topic. They can be found in PDF format at http://www2.hp.com/HP-COMP/rf/app_index.html.

(Imp) Impellimax Application Notes. http://www.impellimax.com/APPNOTES.html. Makers of PIN diode drivers.

(Hor) Horowitz, P., Hill, W., *The Art of Electronics*. 2nd ed. Cambridge UP. (1989)

(Kir95) Kirsch, J., Newman, R., "A pulse NMR experiment for an undergraduate physics laboratory." *Am. J. of P.* preprint.

(Mag) Maguire, Y. *A tabletop quantum computer* (tentative title). MS Thesis. MIT Media Lab. Physics and Media Group. Summer 1999.

(Sen) Senturia, S. Personal communication.

(Shimnote) (summarized from (Chm90) How does one diagnose the error syndrome for an incorrectly shimmed magnet? *For zonal shims*: one can shim zonally by picking a peak in the spectrum and observing its responses to the various shim currents. First, change the current flowing through the (1, 0) shim (the *z* shim, in most spectrometer manuals). As the *z* current changes, if the center frequency of a peak in the spectrum varies, then the *z* shim is not centered on the sample; in this case, the sample may require repositioning. (Accurately determining the centers of the sample and of the shim coil is essential to preserving orthogonality between shims.) After maximizing the sharpness of the peak by varying the (1,0) shim, change the (2, 0) shim (colloquially known as the z^2 shim), until the peak appears to be symmetric. To understand this, note that the amplitude of the additional magnetic field (and thus the Larmor frequency ω_L) varies as the square of *z*. One often sees a 'shoulder' in the spectrum, where the signal falls off due to absence of any sample in magnetic fields of the appropriate strength (due to finite sample size). Then vary the (3, 0) shim: if mistuned, one gets a shoulder plus an inflection point in the spectrum! In real life, the 'odd' *z* shims ((1, 0), (3, 0),...) can cause substantial amounts of lower-order spherical harmonics, while the 'even' shims commonly possess an offset

which shifts the magnetic field; this affects the order in which an experienced shimmer performs the adjustment operations.

For tesseral shims: for a given *dV* at (r, θ, ϕ), the Larmor frequency shift due to the (l, *m*) component if the magnetic field goes like $\Delta w = C_{lm} \cos(m(\phi - \psi_{lm})$. Spinning the sample can immeasurably assist with the shimming process, since the time modulation of the FID indicates what degree *m* of shimming needs modification: the *m*th spherical harmonic is visible as a sideband in the spectrum. In general, the zonal shim effects can be seen in the envelope of the FID, while the tesseral shim effects can be seen in the modulation of the signal emitted by a spinning sample.

(Tag) Fletcher, R. *A Low-cost electromagnetic tagging technology for wireless identification, sensing, and tracking of objects.* MS thesis, MIT Media Lab. (1997)

Digital design (6)

(Arn) Arnold, M. *Verilog digital computer design*. Prentice Hall. (1999)

(Arm) Armstrong, J., *Structured Logic Design With VHDL*. Prentice Hall. (1993)

(Bri) BrightStar Engineering web site, http://www.brightstareng.com

(HP) Hewlett-Packard Test and Measurement Reference Guides for the HPE1437A and HPE1445A. Electronic versions available at
http://www.tmo.hp.com/tmo/datasheets/English/HPE1437A.html
http://www.tmo.hp.com/tmo/datasheets/English/HPE1445A.html

(Xes) XESS documentation. http://www.xess.com/

(Xil) Xilinx Foundation Tools Documentation, F1.5. January 1999.

Mixed-signal design (2)

(AN202) Brokaw, A. P. "An I.C. Amplifier User's Guide to Decoupling, Grounding, and Making Things go Right for a Change." *Analog Devices Application Note AN-202*. Analog Devices, Inc.

(AN342) Brokaw, A. P. "Analog Signal-Handling for High Speed and Accuracy." *Analog Devices Application Note AN-342*. Analog Devices, Inc.

Software design (1)

(Pre) Press, W. et. al. *Numerical Recipes in C, 2nd ed.* (1992)