Factoring in a dissipative quantum computer

César Miquel,[1] Juan Pablo Paz,[1,3] and Roberto Perazzo[1,2]
[1]*Departamento de Física, Facultad de Ciencias Exactas y Naturales, Pabellón 1, Ciudad Universitaria, 1428 Buenos Aires, Argentina*
[2]*Centro de Estudios Avanzados, Universidad de Buenos Aires, J.E. Uriburu 850, 1424 Buenos Aires, Argentina*
[3]*Instituto de Astronomía y Física del Espacio, CC 67, Suc. 28, 1428 Buenos Aires, Argentina*
(Received 26 February 1996)

We describe an array of quantum gates implementing Shor's algorithm [in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, edited by S. Goldwasser (IEEE Computer Society, Los Alamitos, CA, 1994), p. 116; (unpublished); Phys. Rev. A **53**, R2493 (1995)] for prime factorization in a quantum computer. The array includes a circuit for modular exponentiation with several subcomponents (such as controlled multipliers and adders) that are described in terms of elementary Toffoli gates. We present a simple analysis of the impact of losses and decoherence on the performance of this quantum factoring circuit. For that purpose, we simulate a quantum computer that is running the program to factor $N=15$ while interacting with a dissipative environment. As a consequence of this interaction, randomly selected quantum bits (qubits) may spontaneously decay. Using the results of our numerical simulations, we analyze the efficiency of some simple error correction techniques. [S1050-2947(96)03409-9]

PACS number(s): 03.65.Bz

I. INTRODUCTION

In recent years there has been an explosion of activity in the area of quantum computation (see [1,2]). In part, this was a consequence of a very important discovery made by Shor, who demonstrated that two problems, thought to be classically intractable (finding prime factors and discrete logarithms of integer numbers), could be efficiently solved in a quantum computer [3,4]. Shor's results added a practical motivation for the study of quantum computation, which, until that time had received the attention of a smaller community of people interested in fundamental aspects of quantum mechanics, the physics of information, algorithmic complexity theory, etc. By now, quantum computation is a growing field that is developing not only due to the work of theorists but, fortunately, also due to recent advances in experimental techniques. In fact, in recent years there have been a few interesting experiments aiming at constructing quantum gate prototypes (see [5–7]).

There are many open questions concerning the mathematics and also the physics of quantum computers. In fact, we still do not know what the real power of quantum computation is from the algorithmic complexity point of view. (Until now, attempts towards demonstrating their usefulness to solve other nonpolynomial problems were not successful.) On the other hand, the physics of quantum computers also presents many important challenges. Among the most important open questions is the understanding of the impact of the process of decoherence (an issue that has attracted some attention over the past few years [8–12]). Decoherence [13] is a physical process by which the quantum interference effects, essential for the proper functioning of the quantum computer, are destroyed by the interaction between the computer and its environment. This interaction creates irreducible computer-environment correlations, which, as the environment is unobserved, induce the dynamical collapse of the computer's wave function. Decoherence may be potentially devastating, but, as recent studies suggest, there may be ways in which one can reduce the problem. For that purpose, a few ideas have been advanced. Shor proposed a procedure for recovering quantum coherence by using coding [14] (see also [12]), and similar methods have been proposed for ''purifying'' entangled pairs before using them for transmiting quantum information through noisy channels [15]. This, combined with the possibility of building error correction schemes based on the ''watchdog'' effect [16], is a promising idea that is currently under investigation [17–20].

However, to give a specific answer to the question of how important decoherence is for factoring one needs to be rather specific. The answer will depend upon the computer implementation (hardware) and also on the particular algorithm (software) used. For example, the possibility of implementing error correction schemes based on the watchdog effect depends upon having a computer evolving in such a way that at some known instants it is in a known state [or at least some quantum bits (qubits) are in a known state, so that we can measure them without disturbing the computer]. The aim of this paper is to begin a study on the impact of dissipation and decoherence on a quantum factoring computer. For this purpose we design a quantum factoring circuit analyzing how its performance is affected when the interaction with an environment is included.

Several recent papers are related to ours. Chuang *et al.* [10] described on general grounds the potentially devastating effects that decoherence may have upon a factoring computer. Their results, which were obtained using a simple description of the quantum computer, which makes no reference to a specific quantum circuit, suggest that by having a low enough decay rate and using appropriate error correction techniques one may be able to implement factoring in a quantum computer. Cirac and Zoller [5] presented a numerical study of the effects of errors on the quantum Fourier transform (FT) subroutine, which plays a central role in the factoring program. Their simulation was done by considering the effect of spontaneous decay while a computer made of cold trapped ions runs the FT program (designed by Coppersmith and others [4,21]). Other studies of decoherence on quantum computers have been presented that are not directly

related to the issue of factoring. For example, the importance of losses and decoherence have been analyzed [22] for the optical quantum computer designed by Chuang and Yamamoto [23] to solve Deutsch's problem [24] for a one-bit function. The effect of decoherence upon a static quantum computer was also analyzed in [8,11].

The paper is divided in two parts. We first present an array of reversible quantum logic gates that implements Shor's algorithm for factoring integer numbers in a quantum computer. To do that we first created subcomponents that perform some specific tasks such as controlled multiplication, controlled sums, and mod$N$. Then we combined these subcomponents in the precise way required to run Shor's algorithm. The existence of work qubits (required to handle the reversible logic) makes the design of the quantum circuit a rather nontrivial task. In fact, for the quantum computer to work properly, it is necessary to reversibly erase the records created along the computational path (stored in the work qubits). As an example, we present the gate array that could be used to factor $N=15$ in a quantum computer.

Designing the factoring circuit is the first step required for studying the impact of decoherence and the possibility of implementing error correction schemes. This is the purpose of the second part of the paper, where we study how the coupling to an environment affects the functioning of the quantum factoring circuit. For this, we use an oversimplified model of the system-environment interaction. We assume that this interaction takes place only at certain (randomly chosen) moments of time affecting only a few (randomly chosen) qubits, which may spontaneously decay.

After completing the design of the factoring circuit and while we were working on the numerical simulations to model dissipation, we became aware that a very similar gate array was recently developed by Vedral, Barenco, and Ekert [25]. Our circuit produces the same final quantum state and has roughly the same requirements (in number of qubits and time steps) as the one described in [25] (in that paper the authors did not attempt to analyze the impact of losses and decoherence on the performance of their quantum circuit, an issue that we analyze here). More recently Plenio and Knight [26] used some of the conclusions of [25] (the number of required qubits and time steps) to discuss some of the limitations imposed by decoherence on the size of the numbers one could factorize using various physical setups.

In Sec. II we briefly describe both the mathematical basis for Shor's algorithm and the basic steps a quantum computer would need to follow in order to implement it. In Sec. III we describe the quantum network for implementing modular exponentiation. We go from the coarser description, where the circuit is just a black box, to the fine-grained picture, where every component is dissected and built from elementary Toffoli gates. We analyze the architecture required to factor numbers of L bits and explicitly exhibit the circuit to factor $N=15$, that requires 27 qubits (the circuit to factor L bit numbers needs $5L+7$ qubits and involves a number of elementary gates that, for large L, is close to $240L^3$). In Sec. IV we address the importance of decoherence and the possible strategies for error correction. We summarize our results in Sec. V.

II. SHOR'S ALGORITHM

Shor invented an algorithm for a quantum computer that could be used to find the prime factors of integer numbers in polynomial time. We will now briefly review the most important aspects of Shor's algorithm and later consider the way to implement it in a quantum computer.

The mathematical basis for Shor's algorithm is the following (see [3,4,27]). The goal is to find the prime factors of an integer number N. Instead of doing this directly, the algorithm finds the *order* r of a number x. The order of x is defined as the least integer r such that $x^r \equiv 1 \pmod{N}$. Knowing r, one can find the prime factors of N by using some results proved in number theory. Factorization reduces to finding r if one uses a randomized algorithm: as Shor shows in [4], choosing x at random and finding its order r, one can find a nontrivial factor by computing a, the greatest common divisor between $x^{r/2}-1$ and N. In fact, a is a nontrivial factor of N unless r is odd or $x^{r/2}=-1 \bmod N$. As x is chosen at random, the probability for the method yielding a nontrivial prime factor of N is $1-1/2^{k-1}$, where k is the number of distinct prime factors of N.

In his seminal work [3,4], Shor showed that a quantum computer could efficiently find the order r of the number x and therefore factorize N in polynomial time. Let us now describe the basic operation of this quantum computer. This requires two quantum registers, which hold integers represented in binary notation. There should also be a number of work qubits, which are required along the calculation but should be in a standard state (say $|0\rangle$) both at the beginning and at the end of the calculation. The role of these work qubits is very important and will be described in detail in Sec. III. For the moment, we will concentrate on describing the state of the computer before and after every major step of the program. For that purpose, we can forget these qubits for the moment. Apart from the quantum registers, there is also some classical information we should provide for operating the quantum computer. Thus we will assume that the numbers N (the one we want to factor), x (chosen randomly mod$N$), and a randomly chosen q, which is such that $N^2 \leqslant q \leqslant 2N^2$ are part of the classical information available to the quantum computer.

We start the process by preparing the first register in a uniform superposition of the states representing all numbers $a \leqslant q-1$ [this can be done by a standard technique, i.e., rotating each individual qubit, putting it in a superposition $(1/\sqrt{2})(|0\rangle+|1\rangle)$]. The state of the computer is then

$$|\Psi_0\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|0\rangle. \tag{1}$$

The next step is to unitarily evolve the computer into the state

$$|\Psi_1\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|x^a (\bmod N)\rangle.$$

The next step is to Fourier transform the first register. That is, we apply a unitary operator that maps the state $|\Psi_1\rangle$ onto

$$|\Psi_2\rangle=\frac{1}{q}\sum_{a=0}^{q-1}\sum_{c=0}^{q-1}\exp(2\pi iac/q)|c\rangle|x^a(\mathrm{mod}N)\rangle.$$

The final step is to observe both registers (the method could be implemented observing just the first register, but, following Shor [4], for clarity we assume both registers are observed). The probability for finding the state $|c\rangle|x^k(\mathrm{mod}N)\rangle$ is

$$P(c,x^k(\mathrm{mod}N))=\left|\frac{1}{q}\sum_{a/x^a=x^k}\exp(2\pi iac/q)\right|^2,$$

where the sum is over all numbers $0\leq a\leq q-1$ such that $x^a=x^k(\mathrm{mod}N)$. This sum can be transformed into

$$P(c,x^k(\mathrm{mod}N))=\left|\frac{1}{q}\sum_{b=0}^{[(q-1-k)/r]}\exp(2\pi ib\{rc\}_q/q)\right|^2, \tag{2}$$

where $\{rc\}_q$ is an integer in the interval $-q/2<\{rc\}_q\leq q/2$, which is congruent to rc $(\mathrm{mod}q)$. As shown by Shor, the above probability has well defined peaks if $\{rc\}_q$ is small (less than r), i.e., if rc is a multiple of q ($rc=dq$ for some $d<N$). Thus, knowing q and the fact that the position of the peaks c will be close to numbers of the form dq/r, we can find the order r (using well-established continuous fraction techniques).

There is no doubt that Shor's algorithm would work if a quantum computer could be built. However, to implement Shor's algorithm in a quantum computer one needs to explicitly construct the program. The procedure for Fourier transforming is well known and has been extensively discussed in several recent papers (see [4,21,27]). To explicitly construct the unitary evolution that takes the state $|\Psi_0\rangle$ into the state $|\Psi_1\rangle$ is a rather nontrivial task, which we will describe in the next section [25].

III. QUANTUM NETWORK FOR MODULAR EXPONENTIATION

We will present an array of quantum gates that maps the state $|a\rangle\otimes|0\rangle$ onto $|a\rangle\otimes|x^a(\mathrm{mod}N)\rangle$, transforming the state $|\Psi_0\rangle$ into $|\Psi_1\rangle$. We describe the quantum circuit using diagrams such as the one in Fig. 1, which must be interpreted as representing the time evolution of the system with time flowing from left to right. Each line represents a single qubit, i.e., a two-level system (a thick line will represent a bundle of qubits). In describing the circuit we will go in steps from the coarse description of Fig. 1(a) (where the computer is a black box) to a fine-grained description where the computer consists of a complex array of interconnected elementary gates.

We will use Toffoli gates as ''elementary'' components and follow the notation of [28], denoting a gate acting on three qubits as Λ_2. The action of a Toffoli gate on a computational state $|x_1,x_2,x_3\rangle$, (where $x_i\in\{0,1\}$) is $\Lambda_2|x_1,x_2,x_3\rangle=|x_1,x_2,x_3\oplus(x_1\wedge x_2)\rangle$, where $\oplus$ denotes the exclusive OR and $\wedge$ the AND operation between the Boolean variables x_i. Thus Toffoli gates are just controlled-NOT gates where the last qubit changes its state only if the two control qubits are set to 1. It will also be convenient to use

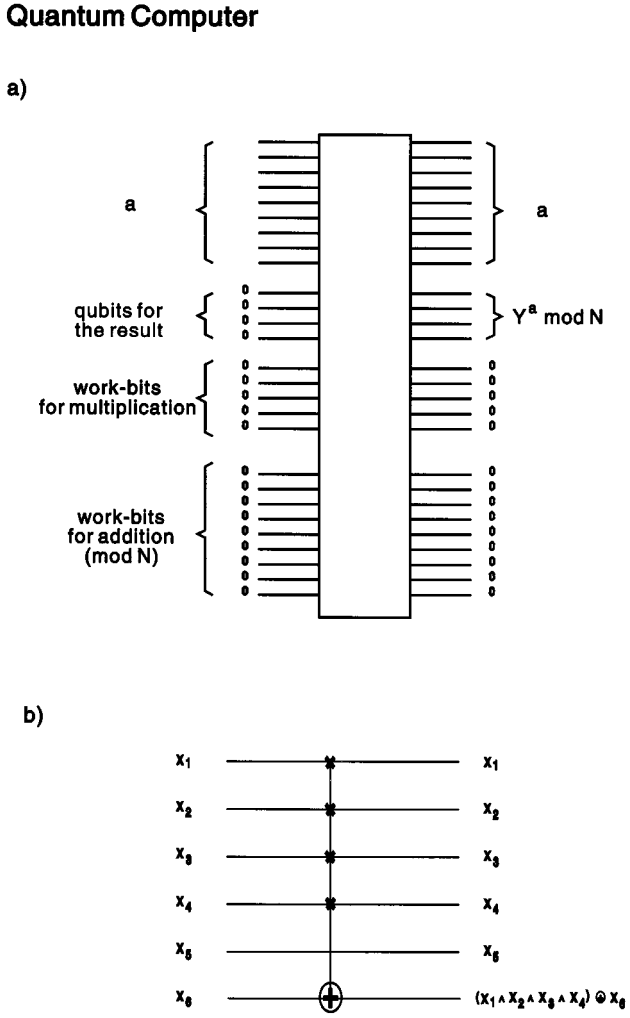

FIG. 1. (a) Black box description of the circuit for modular exponentiation. When N has four bits one needs nine qubits to represent a and fifteen extra qubits to be used as work space. (b) Λ_4 Toffoli gate with four-control bits x_1,x_2,x_3, and x_4. $x_5\rightarrow x_5\oplus(x_1\wedge x_2\wedge x_3\wedge x_4)$.

generalized Toffoli gates, with n control qubits, which are denoted as Λ_n. Of course, all these gates can be constructed in terms of one- and two-qubit operations, as explained in [28]. The diagram representing the gate Λ_n is shown in Fig. 1(b).

To design a quantum circuit for modular exponentiation we should first notice that if the binary representation of a is $a=\Sigma_{i=0}^{n}a_i2^i$, then

$$y^a(\mathrm{mod}N)=\prod_{i=0}^{n}[(y^{2^i})^{a_i}(\mathrm{mod}N)]. \tag{3}$$

Thus modular exponentiation is just a chain of products where each factor is either equal to 1 if $a_i=0$ or equal to y^{2^i} if $a_i=1$. Therefore, the circuit is easily constructed if one is allowed to use a controlled multiplier as an auxiliary unit (which, at this level, acts as a new black box). In Fig. 2 we show the basic architecture of the array of controlled multipliers required for modular exponentiation. For the first multiplication the control qubit is a_0 and after each multiplica-

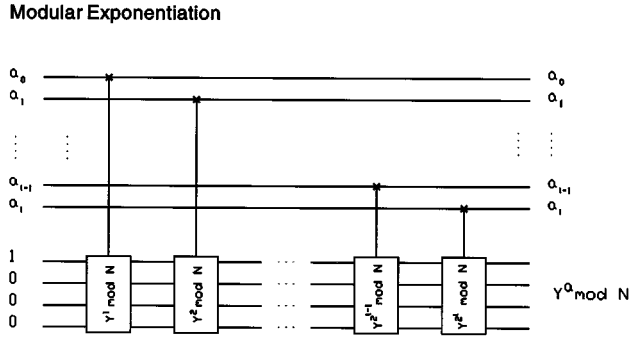

FIG. 2. Gate array used for modular exponentiation. $Y^a \mathrm{mod} N$ is calculated by repeatedly multiplying the second register by $Y^{2^m} \mathrm{mod} N$ only if $a_m = 1$. Each box multiplies its input by $Y^{2^m} \mathrm{mod} N$ only if the control bit a_m is 1.

tion the control is moved to the next qubit. For this array to work we need to know all the numerical factors entering in (3) [thus we must classically compute the numbers $y^{2^i} (\mathrm{mod} N)$].

Our next step is to analyze the controlled multiplier. Given an input $|I\rangle$, this circuit, which we denote as $\Pi_N(C)$, produces an output $|I*C(\mathrm{mod} N)\rangle$. The controlled multiplier is constructed using a smaller black box: a controlled mod$N$ adder. In fact, multiplication of two numbers $I = \Sigma_{i=0}^{n} I_i 2^i$ and C reduces to a sum of the form $\Sigma_{i=0}^{n} I_i (2^i C)$. Thus we just need to use I_i as the control qubit in a controlled mod$N$ adder adding the number $(2^i C)$ [a circuit that we denote as $S_N(2^i C)$]. The numbers involved in the sum must also be provided as classical information (we need to classically compute all numbers $2^j y^{2^i}$, with $i,j \leq L$, where L is the number of bits of N). In Fig. 3 we show a controlled multiplier for four-bit numbers. The same architecture can be used to multiply L-bit numbers. In that case, the controlled multiplier requires $L+1$ work qubits, whose state is set to zero before and after its operation. As we will see below, the controlled adder itself also requires some work space that must be independent of the one used specifically for multiplication.

As shown in Fig. 3, $\Pi_N(C)$ is schematically divided into three pieces. In all of them the work qubits play an important role. The quantum state entering the circuit is $|\chi_0\rangle = |I\rangle \otimes |0\rangle_{\mathrm{WB}}$, where I is the number stored in the input register and $|0\rangle_{\mathrm{WB}}$ is the state of the work qubits. The qubits $|I_i\rangle$ are used as control for the $S_N(2^i C \mathrm{mod} N)$ adders and the result of the sum is temporarily written in the work qubits. After this, the state is $|\chi_1\rangle = |I\rangle \otimes |IC\rangle_{\mathrm{WB}}$: almost what we need, except for the fact that the input $|I\rangle$ also appears in the output state. Erasing this extra copy of the input is essential: Otherwise we would be keeping a record of the computational path affecting the interference pattern of the quantum computer (apart from forcing us to use an enormous amount of space). The reversible erasure of the input is the purpose of the second part of the circuit. In designing this we followed well-known techniques developed by Bennett [29] and described by Shor [4]. The procedure is as follows. We first consider the evolution operator $\tilde{U}$ mapping the input

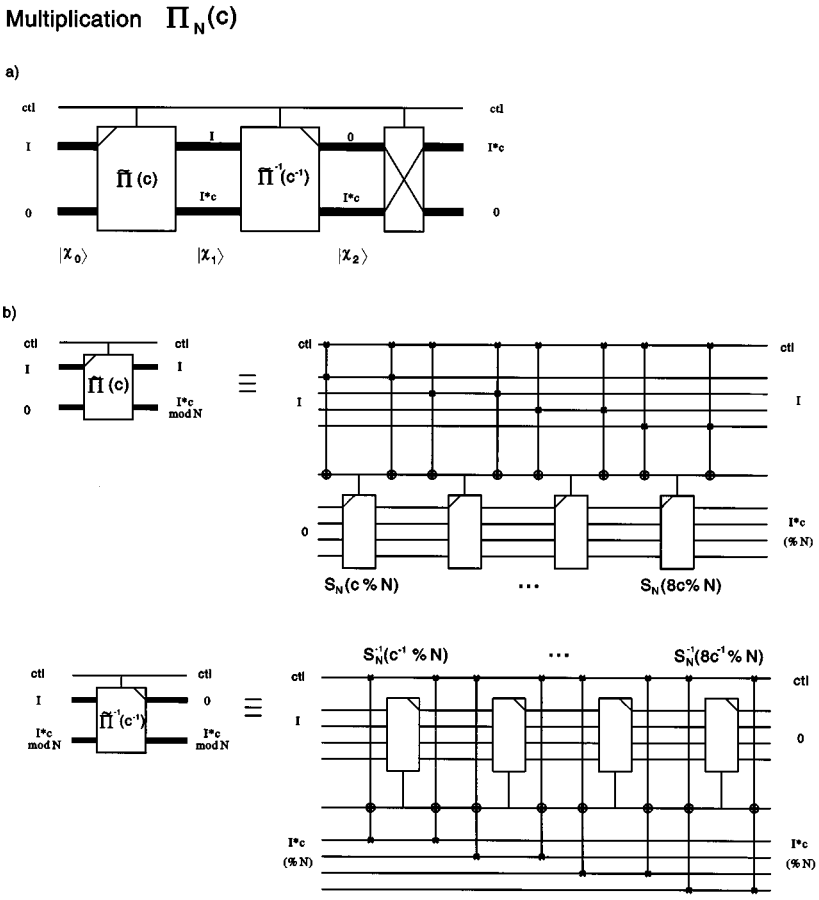

FIG. 3. (a) Three stages of the controlled multiplier (mod N) $\Pi_N(C)$. First the input I is multiplied by C. Then I is reversibly erased and finally the result is swapped with the upper register. (b) Multiplication by C is achieved by repeated addition of $2^m C \mathrm{mod} N$ controlled by I_m. This is done using the controlled mod N adders $S_N(2^m C \mathrm{mod} N)$. In the figure we denote mod$N$ as $\%N$.

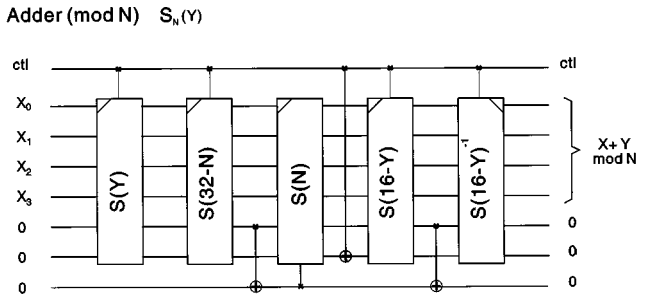

FIG. 4. Addition mod$N$ is achieved with five controlled adders: The first adds C to the input. The second ''subtracts'' N from $a+C$. The third operation adds N only if $a+C$ is smaller than N. At this stage the first four bits have $a+C \bmod N$. The last two stages erase the record left in the seventh bit, whose state depends on the sign of $a+C-N$.

$|0\rangle\otimes|I'\rangle_{\mathrm{WB}}$ onto $|I'C^{-1}\rangle\otimes|I'\rangle_{\mathrm{WB}}$, where C^{-1} is the multiplicative inverse of $C(\mathrm{mod}N)$ [the number satisfying $C*C^{-1}=1(\mathrm{mod}N)$]. The operator needed in the second part of the multiplier is $\widetilde{U}^{-1}$. To convince ourselves that this is the case, we should notice that, as the input to the second part of the multiplier is $|\chi_1\rangle=|I\rangle\otimes|IC\rangle_{\mathrm{WB}}$, the output will be $|\chi_2\rangle=\widetilde{U}^{-1}|\chi_1\rangle=|0\rangle\otimes|IC\rangle_{\mathrm{WB}}$ [because, by construction, $\widetilde{U}$ satisfies $\widetilde{U}(|0\rangle\otimes|IC\rangle_{\mathrm{WB}})=|I\rangle\otimes|IC\rangle_{\mathrm{WB}}=|\chi_1\rangle$]. The circuit for $\widetilde{U}^{-1}$, shown in Fig. 3, is just the specular image of the one used for the first part of the multiplier (switching the role of register and work qubits). Finally, the multiplier is completed with a controlled swap that interchanges once more the register and work qubits so that the final state of the work qubits is always $|0\rangle_{\mathrm{WB}}$.

The circuit for doing controlled mod$N$ sums of a number X, which is stored in a quantum register, and a number Y, stored in a classical register, is called $S_N(Y)$. This circuit, for five-bit numbers, is shown in Fig. 4 (generalization to L-bit numbers is straighforward). The circuit for $S_N(Y)$ is built using a simple controlled adder, which we denote as $S(Y)$, whose functioning will be explained below. The only difference between $S_N(Y)$ and $S(Y)$ is that the former gives the output modulo N. Constructing a reversible circuit for computing the sum mod$N$ is not a trival task, which is only possible because we know that the two numbers being added (X and Y) are both less than N (and therefore $X+Y\leqslant 2N-2$). Without this information it would not be possible to compute mod$N$ reversibly without keeping unwanted records of the computation (since mod$N$ is not a one-to-one function). The input to the circuit is $|\overline{\chi_0}\rangle=|X\rangle\otimes|0\rangle_{\mathrm{WB}}$. After the first adder, this is transformed to $|\overline{\chi_1}\rangle=|X+Y\rangle\otimes|0\rangle_{\mathrm{WB}}$. We then apply another simple adder which adds the positive number $2^{L+1}-N$, thus producing an output $|\overline{\chi_2}\rangle=|2^{L+1}+X+Y-N\rangle\otimes|0\rangle_{\mathrm{WB}}$. The most significant bit (MSB) of $2^{L+1}+X+Y-N$ is one (zero) if $X+Y\geqslant N$ ($X+Y<N$). It is easy to realize that the opposite is true for the second MSB of the output. Thus, if we use this qubit to control the inverse operation, we will add N only if $X+Y<N$. Therefore, after the third gate of the circuit shown in Fig. 5, the first L qubits of the output always store the number $A+C$ mod$N$. However, the $L+1$ and $L+2$ qubits, which are used to control the third gate, keep a record of the first result. As usual, this record must be reversibly

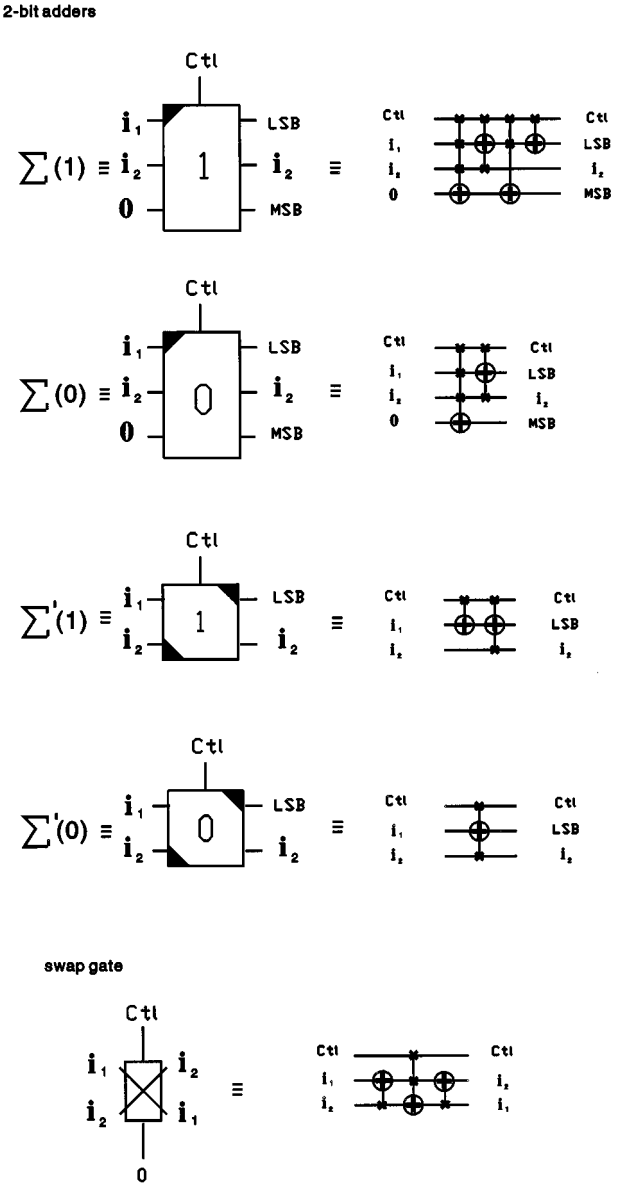

FIG. 5. Two-qubit adders $\Sigma(\sigma)$ are shown in terms of Toffoli gates. They have four input and four output qubits. If the inputs are ctl, i_1, i_2, and 0, the outputs are ctl, the least significant bit (LSB) of $i_1+i_2+\sigma$, i_2, and the most significant bit (MSB) of the sum. A swap gate is also shown that interchanges its two input qubits i_1 and i_2.

erased and this can be done by using the following simple trick: We first add the positive number 2^L-Y and notice that the MSB of the result $2^L-Y+(X \bmod N)$ is always identical to the qubit used to control the third gate. Thus we are done: We apply a control-NOT gate and then we undo the first sum (by adding Y).

So far, we first explained modular exponentiation in terms of controlled multiplication $\Pi_N(C)$. Later, we explained $\Pi_N(C)$ in terms of controlled mod$N$ sums $S_N(Y)$ and this circuit in terms of a simple adder $S(Y)$. We will now present the gate array for the simple controlled adder $S(X)$, which is best explained in terms of a smaller gate: a controlled two-qubit adder. This will be our smallest black box and, for clarity, we will explain here how it works. The two-qubit

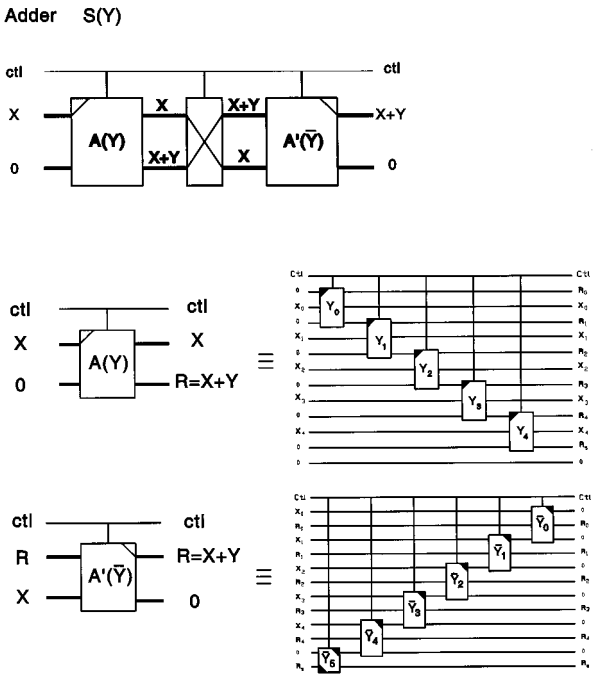

FIG. 6. (a) Addition is performed in three stages: The first adds Y to the input X, the second interchanges X with $X+Y$, and the last reversibly erases the input X. (b) The first and last stages are shown in terms of the individual qubits and two-qubit adders $\Sigma(\sigma)$. Y_0–Y_4 are the bits in the binary representation of Y. $\bar{Y}\equiv 2^L-Y$ is used to erase X.

adder, denoted as $\Sigma(\sigma)$, has four input qubits and a classical input bit σ (i.e., there are two types of two-qubit adders, one for $\sigma=0$ and another for $\sigma=1$). The first input qubit is the control, the second qubit is i_1, the third one is i_2, and the fourth one is a work qubit that is always set to 0 at the input. At the output, the control qubit is unchanged, the first qubit changes into the least significant bit (LSB) of the sum $(i_1+i_2+\sigma)$, the third one stores i_2, and the fourth stores the MSB of the sum. In Fig. 5 we can see how to build the gates $\Sigma(0)$ and $\Sigma(1)$ (and other useful simple gates) in terms of Toffoli gates.

Using $\Sigma(\sigma)$ it is possible to construct a circuit mapping an input $|X\rangle$ into $|X+Y\rangle$. This is displayed in Fig. 6, where, for simplicity, we assumed that both X and Y have five bits. For numbers of L bits the number of work qubits required is $L+3$. The quantum state entering the adder is $|\tilde{\chi}_0\rangle=|X\rangle\otimes|0\rangle_{\rm WB}$. This goes through the sequence of two-qubit adders $\Sigma(Y_i)$ (we use $X_i,Y_i\in\{0,1\}$ for the binary representation of X and Y). After this chain of Σ gates, the state is $|\tilde{\chi}_1\rangle=|X\rangle\otimes|X+Y\rangle_{\rm WB}$, which has an unwanted copy of the input. To reversibly erase this extra copy we apply the same method used in the multiplication. We first consider an auxiliary operator W that adds the positive number $\bar{Y}\equiv 2^L-Y$ ($\bar{Y}$ is known as the 2's complement of Y and its binary representation is simply obtained from that of Y by interchanging zeros and ones and adding 1). The operator W satisfies $W(|R\rangle\otimes|0\rangle_{\rm WB})=|R\rangle\otimes|R+2^L-Y\rangle_{\rm WB}$. Therefore, its inverse is such that $W^{-1}|X+Y\rangle\otimes|2^L+X\rangle_{\rm WB}=|X+Y\rangle\otimes|0\rangle_{\rm WB}$, which is precisely what we need as the output of our circuit (the properties of W^{-1} simply follow from that of W, which, by construction, satisfies $W|X+Y\rangle\otimes|0\rangle_{\rm WB}=|X+Y\rangle\otimes|2^L+X\rangle_{\rm WB}$). Therefore, using W^{-1} after appropriately interchanging the role of the register and the work qubits (and adding an extra work qubit to store the qubit representing 2^L) we complete the controlled adder. The circuit for W^{-1} that is shown in Fig. 6 is almost the specular image from the one used as the first part of the adder. The only difference is that instead of the first two-qubit adder we can use a smaller circuit that only stores the LSB of the first sum (this circuit is shown in Fig. 5).

Having explained the essential pieces of the quantum computer, let us now summarize what its space and time requirements are (i.e., the number of qubits and the number of elementary operations). As explained above, to factor an L-bit number we need $L+1$ qubits as work space for the controlled multiplier and $L+4$ for controlled sums. The mod$N$ circuit requires an extra qubit. Adding the qubits required to store the two quantum registers ($2L+1$ qubits to store a in the first register and L qubits for the second register) we get a total of $5L+7$ qubits. Computing the number of elementary operations is also possible. By inspecting our controlled adder one realizes that the number of elementary gates is $\alpha L+\beta(L+1)$, where α and β are, respectively, the number of gates in a two-qubit adder and in its inverse and the one in a swap circuit. Using the estimate $\alpha=\beta=3$ one gets $12n+17$ operations for the sum. Using similar arguments to analyze the multipliers one finally concludes that the complete modular exponentiation circuit requires $240n^3+484n^2+182n$ elementary operations. For $L=4$ this is about 2.5×10^4.

IV. LOSSES AND DECOHERENCE IN A FACTORING COMPUTER

Before analyzing the impact of dissipative effects on the quantum circuit it is convenient to introduce some notation. The quantum computer has a Hilbert space with a computational basis with states $|r_1,r_2,\mathcal{W}\rangle$ (where r_1, r_2, and $\mathcal{W}$ are the bit strings determining the states of the first register, the second register, and the work qubits, respectively). We assume that the environment $\mathcal{E}$ has a Hilbert space spanned by a basis of states $|e\rangle_{\mathcal{E}}$. The quantum state of the computer–environment ensemble can always be written as

$$|\Psi(t)\rangle=\sum_{r_1,r_2,\mathcal{W},e}A(r_1,r_2,\mathcal{W},e,t)|r_1,r_2,\mathcal{W}\rangle|e\rangle_{\mathcal{E}}. \quad (4)$$

The temporal evolution of the probability amplitude $A(r_1,r_2,\mathcal{W},e,t)$ is governed by the interplay between the quantum circuit described in Sec. III and the computer-environment interaction. At the initial time, when the computer is in state (1), the amplitudes are given by

$$A(r_1,r_2,\mathcal{W},e,t=0)=\frac{1}{\sqrt{q}}\delta(r_2,0)\,\delta(\mathcal{W},0)\,\delta(e,0).$$

Here we assumed that the computer is initially uncorrelated with the environment, which is taken to be in an unexcited state $|0\rangle_{\mathcal{E}}$ [we use $\delta(a,b)$ to denote Kronecker's delta function]. If the computer evolves without interaction with the environment the amplitudes after the modular exponentiation circuit are

$$A_{\text{exact}}(r_1,r_2,\mathcal{W},e,t=t_f)=\frac{1}{\sqrt{q}}\delta(r_2,y^{r_1}(\text{mod}N))\times\delta(\mathcal{W},0)\,\delta(e,0). \quad (5)$$

However, when the computer interacts with the environment, the actual amplitudes will deviate from the exact expression (5). To model this interaction we will use a very simple approach that incorporates the losses induced by the spontaneous decay of the computer's qubits: The environment consists of a collection of two level systems $\mathcal{E}_i$, i.e., a collection of ''environmental qubits'' (each $\mathcal{E}_i$ qubit has an excited state $|1\rangle_{\mathcal{E}_i}$ and a ground state $|0\rangle_{\mathcal{E}_i}$). For simplicity we will assume that at a given time, a randomly selected computer qubit q_i interacts with one of environmental qubits $\mathcal{E}_i$. As a result of this sudden interaction correlations are established according to

$$|1\rangle_{q_i}|0\rangle_{\mathcal{E}_i}\rightarrow p_1^{1/2}|1\rangle_{q_i}|0\rangle_{\mathcal{E}_i}+p_2^{1/2}|0\rangle_{q_i}|1\rangle_{\mathcal{E}_i}, \quad (6)$$

$$|0\rangle_{q_i}|0\rangle_{\mathcal{E}_i}\rightarrow|0\rangle_{q_i}|0\rangle_{\mathcal{E}_i},$$

where $p_2=1-p_1$. The interpretation of the evolution (6) is quite clear. If the computer qubit is in the state $|1\rangle_{q_i}$ it has a probability p_1 to persist and a probability p_2 to decay into $|0\rangle_{q_i}$, creating an excitation in the environment. On the other hand, if the computer qubit is in the state $|0\rangle_{q_i}$ nothing happens. It is worth mentioning that the decay rules (6) implicitly assume that the state used to represent the *computational* 0 is the ground state (or, at least, has lower energy than the one used to represent the *computational* 1). In fact, the situation may be exactly the opposite in which case the rules (6) must be trivially modified by interchanging the roles of $|1\rangle_{q_i}$ and $|0\rangle_{q_i}$ (see below). More general evolution rules (such as the ones used in, [14] which are best suited to analyze a noisy but almost losseless computer) will be studied elsewhere [30].

Thus we can summarize the basic ingredients of our computer-environment model. (i) It is characterized by a randomly chosen sequence of times $(t_1,\ldots,t_n)$ that define the instants where the computer interacts with the environment (in between these times the computer evolves according to the unitary operators associated with the quantum circuit described in Sec. III). (ii) At each time t_i we randomly choose a computer qubit q_i that is involved in a sudden interaction with an environmental qubit $\mathcal{E}_i$. (iii) As a consequence of this interaction the computer-environment ensemble evolves according to the rules (6). Implicit in our assumptions is the validity of the simplifying Markovian approximation, which ensures that at every instant t_i a different (and independent) environmental qubit $\mathcal{E}_i$ is involved in the interaction. A simple way of visualizing this computer-environment model is by thinking of the times t_i as the instants where there may be a ''branching'' of the computational trajectory. Every time an environmental qubit is excited an ''erroneous'' computational trajectory emerges. At the end of the modular exponentiation circuit, the state vector of the computer-environment ensemble is written as in (4) with an amplitude that will *not* be given by (5). We already admitted that this is an oversimplification of reality (which has been used before to model losses in quantum computation [22]).

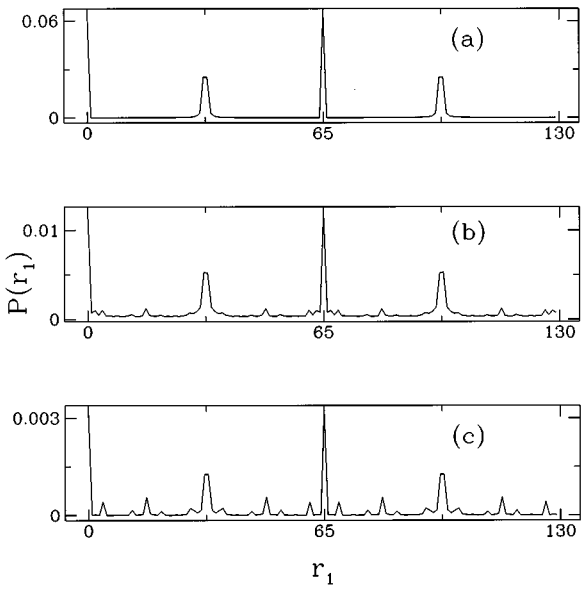

FIG. 7. Probability distribution for r_1 and $r_2=7$. In the simulations $N=15$, $q=130$, and $p_1=p_2=1/2\forall t\in[0,1]$. (a) Exact result. (b) Result with ten decaying qubits at randomly chosen instants of time t_1–t_{10}. (c) Probability distribution for r_1, $r_2=7$, and all work qubits in their zero state.

We computed the amplitudes from the output state of the Fourier transform circuit, which follows modular exponentiation (the discrete FT circuit is described in the literature [4,5,21]). In Fig. 7 results are presented for the probability of finding r_1 in the first register and $r_2=7$ in the second register. The ideal result, plotted in Fig. 7(a), is obtained from Eq. (2). This error-free curve has three sharp peaks, with a separation approximately equal to $q/r=130/4$ (we deliberately choose a rather small value for q so that the small structure in the plots can be seen using a reasonable scale). Provided we do not know the final state of the environment and the work qubits (see below) the probability is

$$P_{\text{NED}}(r_1,r_2)=\sum_{\mathcal{W},e}|A(r_1,r_2,\mathcal{W},e,t)|^2$$

(the subscript NED stands for ''no error detection,'' see below). This probability is shown in Fig. 7(b), where we can see that the errors slightly widen the peaks and notably decrease their amplitudes. As the number of errors is increased it will be less and less likely to measure a value of r_1 located near a peak making the identification of the order r (obtained from the separation between peaks, as explained in [4]) more and more difficult. The appearence of intermediate peaks is also evident in Fig. 7(b). Apart from the above probability we also calculated the probability for finding r_1 in the first register, $r_2=7$ in the second, and the work qubits in the state $|0\rangle_{\text{WB}}$, i.e.;

$$P_{\text{ED}}(r_1,r_2)=\sum_{e}|A(r_1,r_2,\mathcal{W}=0,e,t)|^2.$$

This is plotted in Fig. 7(c), where we see that while a noisy dc component [present in (b)] is supressed, the amplitude ratio between the misleading and correct peaks is increased. These plots correspond to simulations of the quantum computer running the program to factor $N=15$ while coupled to an environment at a randomly chosen set of ten instants t_i (we use $p_1=p_2=1/2$). The modular exponentiation circuit requires about 2.5×10^4 elementary (Toffoli) gates. This roughly correspond to 10^5 one-bit operations for Cirac and Zoller's cold ions computer, [5] Thus, in that case we are considering an error rate of order 10^{-4}, which is a rather optimistic figure.

Our simulations can be used not only to visualize the importance of the environmental interaction on the quantum algorithm but also to test simple error detection (and correction) schemes. The simplest of such schemes is probably the one based on checking the state of the qubits that are supposed to be in a known state. Our factoring program is suited for this purpose since the work qubits must start and end in the state representing the computational 0. Two comments concerning error detection (and correction) are in order. First, by checking the final state of the work qubits we are not able to detect a special class of errors that are produced by the decay of the qubits representing the first and second registers of the computer (r_1 and r_2). Errors of that kind leave (most of the time) the work qubits untouched but generate a misleading output [they are responsible for the intermediate peaks seen in Fig. 7(c), which make the measurement of the order r a much more difficult task]. Second, and more important, by measuring the final state of the work qubits we are only able to *detect* errors but not to correct (or prevent) them.

Of course, it would be much better to have a method enabling us to *prevent* the errors from occurring. For this, the use of the watchdog effect [16] has been proposed. Thus, if some of the computer's qubits are supposed to be in a known state at some time, one could inhibit their decay by making a measurement on the known state. This method can indeed be applied here since the work qubits are supposed to be in the state representing the computational 0 at many intermediate instants of the computation. In fact, this is what happens after the action of each $\Pi_N(C)$ circuit and after the action of each controlled adder $S_N(C)$. For large L, the number of times one could measure the state of some of the work qubits grows as L^2.

To test the efficiency of the watchdog effect as an error correction technique we slightly changed our computer-environment interaction model. In fact, we now assume that the decay rules are of the form (6) but with time-dependent coefficients given by

$$p_1(t)=\begin{cases}\left(1-\dfrac{t^2}{t_0^2}\right)^2, & 0\leqslant t\leqslant t_1\\ A\exp(-2\gamma t), & t>t_1,\end{cases}$$

$$p_2(t)=1-p_1(t). \tag{7}$$

In this way, after having an initial ($t<t_1$) quadratic dependence, the survival probability for a qubit decreases exponentially with time (measured from the start of the computation and, by convention, expressed in units of the total time required to run the program, i.e., $t=1$ corresponds to the end of the computation). In the simulations shown in Fig. 8 we considered $\gamma=1,2$ and $t_1\sim10^{-2}\gamma^{-1}$ so that towards the end of the computation a qubit will have a high decay probability.

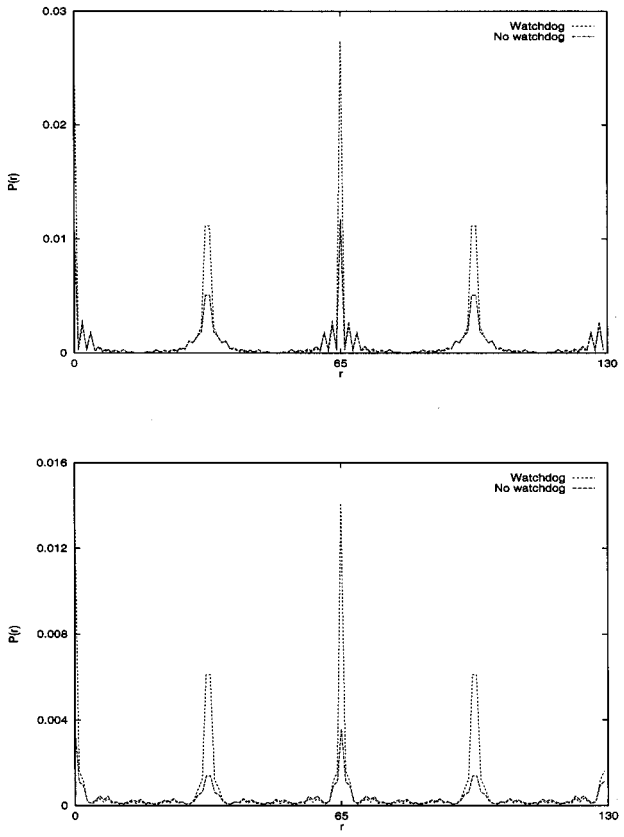

FIG. 8. Graphs showing the probability distribution for r_1 and $r_2=7$ with $q=130$. We simulated the circuit for factoring $N=15$ with ten decaying qubits at randomly chosen instants of time. The probability distribution is plotted with and without using the watchdog effect. The parameters of Eq. (7) are (a) $\gamma=1$, $t_0=0.173$, and $A=1.008$ and (b) $\gamma=2$, $t_0=0.127$, and $A=1.016$.

To implement the watchdog we measure the state of the work qubits at every instant when they are supposed to be in the computational 0. Every time we do this we reset the time in (7). Thus a work qubit will decay with probabilities proportional to the ones given in (7), where the time will effectively be measured from t_{last}, the last instant at which the work qubit was supposed to be in the computational 0 state. Therefore the survival probability for a work qubit is

$$p_{\text{WQ}}(t)=p_1(t-t_{\text{last}})\prod_{i=0}^{n}p_1(\Delta t_i),$$

where Δt_i is the time between two consecutive measurements of the work qubits. On the other hand, the qubits involved in the first or second registers of the computer will have decay probabilities given by (7) with time counting from the beginning of the computation.

The effectiveness of the watchdog effect as an error prevention technique can be seen in Fig. 8, where the probability with and without the watchdog effect are plotted together. Without using this method we get a very noisy probability

with a substantial widening of the principal peaks. The amplitude of the central peak, which in Fig. 8(a) is about 0.01, is of the same order as the one shown in Fig. 7(b) (but the decay rules we are using here are more damaging than the ones we used before). Using the watchdog technique we substantially increase the amplitude of the main peaks (by a factor of 2 or 3) and also eliminate almost all the noise. The only remaining spurious peaks are those produced by the decay of qubits involved in the first and second registers. They cannot be eliminated using the watchdog effect since their existence is not a consequence of a process affecting the work qubits. It is worth stressing that for the watchdog effect to be useful in preventing errors, the Zeno time t_1 must be larger than the time required for a work bit to return to its logical zero state. This is a rather strong constraint and may turn out to be physically unrealistic in most situations.

V. SUMMARY AND OUTLOOK

The factoring circuit we presented is by no means optimal. Several improvements are possible to reduce the number of work qubits. However, when designing a circuit for practical purposes one has to keep in mind that the existence of work qubits is not necesarily a burden. Our results show they can play a very useful role by allowing the use of the watchdog effect as an error prevention technique. It would be important to find the optimal balance minimizing the number of work qubits but still allowing an efficient use of the watchdog method.

The simulations we performed are rather simple and do not allow us to test the importance of other sources of problems for quantum computers. One of the most important sources of errors we excluded here is related to the fact that the elementary quantum gates are never 100% efficient. If we think of Cirac and Zoller's [5] cold ion hardware, the elementary gates are built by applying a sequence of laser pulses on individual ions. If these pulses are not exact π pulses (or $\pi/2$ pulses) the quantum gate will not be exactly the one we want. The corresponding unitary evolution operator U_{real} will have nonzero matrix elements in places where the exact quantum gate operator U_{ideal} has zero matrix elements. These imperfections may be rather important since their effects accumulate in time. To include this effects in our model one needs to follow the evolution of the computer's state vector in the 2^{28}-dimensional Hilbert space. Even though our work enables us to explicitly write down the matrix U_{real} at every step of the calculation, we are not able to numerically simulate this because of space limitations (thus simulating a quantum computer with N qubits needs an exponentially large amount of space in a classical computer). Simulations of smaller versions of our circuit for modular exponentiation are planned to be presented elsewhere [30].

ACKNOWLEDGMENT

We thank Adriano Barenco for useful comments on an earlier version of this manuscript. J.P.P. was partially supported by grants from UBACYT, Fundación Autorchas, and CONICET.

[1] S. Lloyd, Sci. Am. **273**, 44 (1995).
[2] C. Bennett, Phys. Today **48** (10), 24 (1995); C. Bennett and D. DiVincenzo, Nature **377**, 389 (1995).
[3] P. Shor, in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, edited by S. Goldwasser (IEEE Computer Society, Los Alamitos, CA, 1994), p. 116.
[4] P. Shor, SIAM J. Comput. (to be published).
[5] A. Cirac and P. Zoller, Phys. Rev. Lett. **74**, 4091 (1995).
[6] T. Sleator and H. Weinfurter, Phys. Rev. Lett. **74**, 4087 (1995).
[7] Q. A. Turchette, C. J. Hood, W. Lange, H. Mabuchi, and H. J. Kimble, Phys. Rev. Lett. (to be published).
[8] W. G. Unruh, Phys. Rev. A **51**, 992 (1995).
[9] J. P. Paz (unpublished).
[10] I. Chuang, R. Laflamme, P. Shor, and W. Zurek, Science **270**, 1633 (1995).
[11] G.M. Palma, K.-A. Suominen, and A. Ekert (unpublished).
[12] I. Chuang and R. Laflamme (unpublished).
[13] W. H. Zurek, Phys. Today **44** (10), 36 (1991); **46** (4), 13 (1993).
[14] P. Shor, Phys. Rev. A **53**, R2493 (1995).
[15] C. Bennett *et al.*, Phys. Rev. Lett. **76**, 722 (1996).
[16] W. H. Zurek, Phys. Rev. Lett. **53**, 391 (1984).
[17] A. M. Steane, Phys. Rev. A (to be published).
[18] A. R. Calderbank, E. M. Rains, N. J. Sloane, and P. W. Shor (unpublished).
[19] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek, Phys. Rev. Lett. **77**, 198 (1996).
[20] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wooters (unpublished).
[21] D. Coppersmith, IBM Research Report No. RC19642, 1994 (unpublished).
[22] I. Chuang, R. Laflamme, J. P. Paz, and T. Yamamoto (unpublished).
[23] I. Chuang and Y. Yamamoto, Phys. Rev. A **52**, 3489 (1995).
[24] D. Deutsch and R. Josza, Proc. R. Soc. London Ser. A **439**, 553 (1992).
[25] V. Vedral, A. Barenco, and A. Ekert, Phys. Rev. A **54**, 147 (1996).
[26] M. B. Plenio and P. L. Knight, Phys. Rev. A **53**, 2986 (1996).
[27] A. Ekert and R. Josza, Rev. Mod. Phys. (to be published).
[28] A. Barenco *et al.*, Phys. Rev. A **52**, 3457 (1995).
[29] C. Bennett, IBM J. Res. Dev. **17**, 525 (1973); SIAM J. Comput. **18**, 766 (1989).
[30] C. Miquel, J. P. Paz, and R. Perazzo (unpublished).