Digital Switching in the Quantum Domain

I.-Ming Tsai and Sy-Yen Kuo, *Fellow, IEEE*

***Abstract*—In this paper, we present a switching architecture such that digital data can be switched in the quantum domain. The proposed mechanism supports unicasting as well as multicasting, and is strict-sense nonblocking. In addition, with appropriate interface conversion, this architecture can also be used to switch classical information. This results in a quantum switch that can be used to build classical and quantum information networks. To present this idea, we define the connection digraph which can be used to describe the behavior of a switch at a given time, then we show how a connection digraph can be implemented using elementary quantum gates. Compared with a traditional space or time domain switch, the proposed switching mechanism is much more scalable. Assuming an $n \times n$ quantum switch, the space consumption grows linearly, i.e., $O(n)$, while the time complexity is $O(1)$ for unicasting, and $O(\log_2 n)$ for multicasting. Based on these advantages, a high-throughput switching device can be built simply by increasing the number of I/O ports.**

***Index Terms*—Digital switching, quantum circuits, quantum switching.**

I. Introduction

THE demand for bandwidth is rapidly increasing due to the explosive growth of network traffic. Networking technologies play an important role in bridging the gap between limited resources and the constantly increasing demand. In order to avoid a fully meshed architecture, a switching device is required to build a realistic network. Over the past few years, several enabling technologies have emerged as candidates for achieving high performance switching. Basically, switches act like automated patch-panels, switching all the electrical or optical signals from one port to another. Traditionally, digital switching can be done in many ways. For example, by allocating physical separated paths, switching can be done in the space domain. A two–dimensional (2-D) microelectromechanical system (MEMS) optical switch with precisely controlled micromirrors is essentially a space-domain switch. Similarly, by associating the data from each port with a unique resource, switching can be performed in many other ways, such as in the time domain, the wavelength domain, and even a combination of these mechanisms.

On the other hand, quantum information science is a relatively new field of study. Quantum computers were first discussed in the early 1980s [1]–[3]. Since then, a great deal of research has been focused on this topic. Remarkable progress has been made due to the discovery of secure key distribution [4], polynomial time prime factorization [5], and fast database search algorithm [6]. These results have recently made quantum information science the most rapidly expanding research field. Other applications, such as clock synchronization [7], [8], and quantum Boolean circuit implementation [9] have driven this field further into the phase of real-world applications.

In this paper, we present a switching architecture such that digital data can be switched in the quantum domain. The proposed mechanism supports unicasting as well as multicasting, and is strict-sense nonblocking. In addition, with appropriate interface conversion, this architecture can also be used to switch classical information. This results in a quantum switch that can be used to build classical and quantum information networks. To present this idea, we define the connection digraph which can be used to describe the behavior of a switch at a given time, then we show how a connection digraph can be implemented using elementary quantum gates. Compared with a traditional space or time domain switch, the proposed switching mechanism is much more scalable. Assuming an $n \times n$ quantum switch, the space consumption grows linearly, i.e., $O(n)$, while the time complexity is $O(1)$ for unicasting, and $O(\log_2 n)$ for multicasting. Based on these advantages, a high throughput switching device can be built simply by increasing the number of I/O ports.

II. Notations and Preliminaries

A. Quantum State and Quantum Gates

In a two-state quantum system, each bit can be represented using a basis consisting of two eigenstates, denoted by $|0\rangle$ and $|1\rangle$, respectively. These states can be either spin states of a particle ($|0\rangle$ for spin-up and $|1\rangle$ for spin-down) or energy levels in an atom ($|0\rangle$ for ground state and $|1\rangle$ for excited state). These two states can be used to simulate the classical binary logic. A classical binary logic value must be either ON (1) or OFF (0), but not both at the same time. However, a bit in a quantum system can be any linear combination of these two states, so we have the state $|\psi\rangle$ of a bit as

$$|\psi\rangle = c_0|0\rangle + c_1|1\rangle \tag{1}$$

where c_0, c_1 are complex numbers and $|c_0|^2 + |c_1|^2 = 1$. In column matrices, this is written as

$$|\psi\rangle = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}. \tag{2}$$

The state shown above exhibits a unique phenomenon in quantum mechanics called *superposition*. When a particle is in such a superposed state, it has a part corresponding to $|0\rangle$ and a part corresponding to $|1\rangle$, at the same time. When you measure the particle, the system is projected to one of its basis (i.e. either $|0\rangle$ or $|1\rangle$). The overall probability for each state is given by the absolute square of its amplitude. Taking the state $|\psi\rangle$ in (1) as an example, the coefficient $|c_0|^2$ and $|c_1|^2$ represents

Manuscript received July 10, 2002; revised September 25, 2002.

The authors are with the Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: sykuo@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/TNANO.2002.806824

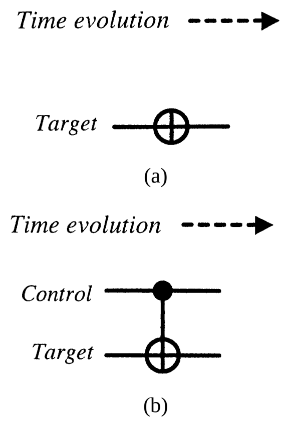


Fig. 1. Symbols for an $\mathbf{N}$ and a $\mathbf{CN}$ gate.

the probability of obtaining $|0\rangle$ and $|1\rangle$ respectively. Obviously, the sum of $|c_0|^2$ and $|c_1|^2$ will be 1 to satisfy the probability rule. To distinguish the above system from the classical binary logic, a bit in a quantum system is referred to as a quantum bit, or *qubit.*

Two or more qubits can also form a quantum system jointly. A two-qubit system is spanned by the basis of the tensor product of their own spaces. Hence, the joint state of qubit A and qubit B is spanned by $|00\rangle_{AB}$, $|01\rangle_{AB}$, $|10\rangle_{AB}$, and $|11\rangle_{AB}$, i.e.,

$$|\phi\rangle_{AB} = c_0|00\rangle_{AB} + c_1|01\rangle_{AB} + c_2|10\rangle_{AB} + c_3|11\rangle_{AB} \quad (3)$$

where c_0, c_1, c_2, c_3 are all complex numbers and $|c_0|^2 + |c_1|^2 + |c_2|^2 + |c_3|^2 = 1$. In matrix form, this is equivalent to

$$|\phi\rangle_{AB} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}. \quad (4)$$

The notations described above can be generalized to a multiple-qubit system. For example, in a three-qubit system, the space is spanned by a basis consisting of eight elements $(|000\rangle_{ABC}, |001\rangle_{ABC}, \ldots, |111\rangle_{ABC})$.

A quantum system can be manipulated in many different ways, called *quantum gates*. A quantum gate can be represented in the form of a matrix operation. For example, a quantum *'Not'* $(\mathbf{N})$ gate applied on a single qubit can be represented by multiplying a 2 × 2 matrix

$$N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (5)$$

which changes the state from $|1\rangle$ to $|0\rangle$ and from $|0\rangle$ to $|1\rangle$, as

$$N \cdot \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_0 \end{pmatrix}. \quad (6)$$

The symbol of an $\mathbf{N}$ gate is shown in Fig. 1(a). Note that the horizontal line connecting the input and the output is not a physical wire as in classical circuits, it represents a qubit under time evolution.

Similarly, a two-bit gate can be represented by a 4 × 4 matrix. For example, a *'Control-Not'* $(\mathbf{CN})$ gate is represented by

$$CN = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \quad (7)$$

The symbol of a $\mathbf{CN}$ gate is shown in Fig. 1(b). A $\mathbf{CN}$ gate consists of one *control* qubit, which does not change its value, and a *target* qubit, which changes its value only if the control qubit is $|1\rangle$. The gate can be written as $CN(|control,\ target\rangle) = |control, control \oplus target\rangle$, where '$\oplus$' denotes exclusive-or. In matrix form, a $\mathbf{CN}$ gate changes the probability amplitudes of a quantum system as follows:

$$CN \cdot \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ c_3 \\ c_2 \end{pmatrix}. \quad (8)$$

A generalization of the quantum gates described above involves *rotation* and *phase shift*. For example, the *'not'* operations on the targets in Fig. 1(a) and (b) can be replaced using general single-bit operations

$$U = \begin{pmatrix} e^{i(\delta+\frac{\alpha}{2}+\frac{\beta}{2})}\cos(\frac{\theta}{2}) & e^{i(\delta+\frac{\alpha}{2}-\frac{\beta}{2})}\sin(\frac{\theta}{2}) \\ -e^{i(\delta-\frac{\alpha}{2}+\frac{\beta}{2})}\sin(\frac{\theta}{2}) & e^{i(\delta-\frac{\alpha}{2}-\frac{\beta}{2})}\cos(\frac{\theta}{2}) \end{pmatrix}. \quad (9)$$

This matrix controls the phase difference and relative eigenstate contributions of the target qubit. Just like AND and NOT form a universal set for classical boolean circuits, one- and two-bit gates are sufficient to implement any unitary operation [10], [11]. A set of quantum gates which can be used to implement any unitary operation is called a universal set. There are many universal sets of one- and two-bit gates. A practical approach is to use general one-bit rotation gates as in (9) and the $\mathbf{CN}$ gate as a universal set.

B. Qubit Permutation and Replication

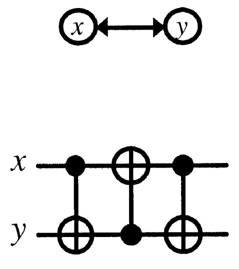
An important property regarding a quantum boolean operation is that any quantum boolean logic can be represented using a *permutation*. A permutation is a one-to-one and onto mapping from a finite-order set onto itself. A typical permutation P is represented using the symbol

$$P = \begin{pmatrix} a & b & c & d & e & f \\ d & e & c & a & f & b \end{pmatrix}. \quad (10)$$

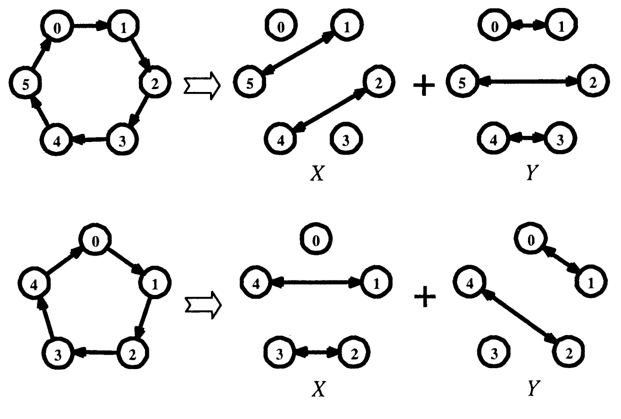
This permutation changes $a \rightarrow d$, $d \rightarrow a$, $b \rightarrow e$, $e \rightarrow f$, and $f \rightarrow b$, with state c remaining unchanged. A permutation can also be expressed as disjoint *cycles*. A cycle is basically an ordered list, which is represented as

$$C = (e_1, e_2, \ldots, e_{n-1}, e_n). \quad (11)$$

The order of the elements describes the operation. For example, in (11), the cycle takes $e_1 \rightarrow e_2, e_2 \rightarrow e_3, \ldots, e_{n-1} \rightarrow e_n$, and finally $e_n \rightarrow e_1$. The number of elements in a cycle is called *length*. A cycle of length 1 is called a *trivial* cycle, which can be ignored as it does not change anything. A cycle of length 2 is



(a)



(b)

Fig. 2. Circuit for (a) a transposition and (b) general cycles.

called a *transposition*. Using this notation, the same permutation P shown in (10) can be written as

$$P = (a, d)(c)(b, e, f) = (a, d)(b, e, f). \tag{12}$$

As we can see, a simple quantum boolean gate like $\mathbf{CN}$ can be regarded as a permutation, because the probability amplitudes in the quantum state are manipulated in the same way. In other words, a quantum Boolean logic gate can be expressed as a permutation, or cycles. For example, a $\mathbf{CN}$ of gate is indicated by $P_{CN} = (10, 11)$, changing 10 $\rightarrow$ 11 and11 $\rightarrow$ 10, leaving all other states unchanged.

In addition to permuting the eigenstates, a qubit can be permuted as a whole. This is equivalent to reshuffling the quantum states for each of the qubits. Since a permutation can be decomposed into disjoint cycles, the implementation actually consists of executing cycles of various lengths in parallel. Because a cycle of length 1 does not permute anything, no circuit is required for a trivial cycle. For a cycle of length 2, the transposition can be done by three $\mathbf{CN}$ gates, as shown in Fig. 2(a).

The circuit in Fig. 2(a) is described as follows. For a two-qubit system

$$|\psi, \phi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle \tag{13}$$

the circuit transforms $|00\rangle \rightarrow |00\rangle$, $|01\rangle \rightarrow |10\rangle$, $|10\rangle \rightarrow |01\rangle$, and $|11\rangle \rightarrow |11\rangle$. This is equivalent to the permutation

$$P = (c_{00})(c_{01}, c_{10})(c_{11}). \tag{14}$$

Assuming the state of these two unentangled qubits are $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\phi\rangle = \gamma|0\rangle + \delta|1\rangle$, where $\alpha, \beta, \gamma, \delta \in C$ and $|\alpha|^2 + |\beta|^2 = |\gamma|^2 + |\delta|^2 = 1$, the joint state

$$|\psi\rangle \otimes |\phi\rangle = \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle \tag{15}$$

is transformed to

$$\begin{aligned} &\alpha\gamma|00\rangle + \beta\gamma|01\rangle + \alpha\delta|10\rangle + \beta\delta|11\rangle && (16)\\ &= (\gamma|0\rangle + \delta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle) && (17)\\ &= |\phi\rangle \otimes (|\psi\rangle && (18)\end{aligned}$$

which does the transposition. Note that once we have this basic function, we can build a switching network in the same way as a classical space switch. However, a more efficient implementation exists, as will be presented later in this paper.

For a general n-qubit ($n \geq 3$) cycle $C = (q_0, q_1, q_2, \cdots, q_{n-1})$, it can be done by six layers of $\mathbf{CN}$ gates without ancillary qubits [12]. The quantum operations required to implement C are described as follows. For an even $n (n = 2m, m = 2, 3 \ldots)$, we define the following nonoverlapping qubit transpositions as:

$$\begin{aligned} X &= (q_{m-1}, q_{m+1}) \cdots (q_2, q_{n-2})(q_1, q_{n-1}) && (19)\\ Y &= (q_m, q_{m+1}) \cdots (q_2, q_{n-1})(q_1, q_0). && (20)\end{aligned}$$

The cycle can be implemented using

$$U = YX. \tag{21}$$

On the other hand, for an odd $n (n = 2m + 1, m = 1, 2, 3 \ldots)$, we define the following nonoverlapping qubit transpositions as:

$$\begin{aligned} X &= (q_m, q_{m+1}) \cdots (q_2, q_{n-2})(q_1, q_{n-1}) && (22)\\ Y &= (q_m, q_{m+2}) \cdots (q_2, q_{n-1})(q_1, q_0). && (23)\end{aligned}$$

Note that if the subscript $m + 2 \geq n$, then $mod(m + 2, n)$ is used to avoid ambiguity. In the same way, the cycle can be implemented using

$$U = YX. \tag{24}$$

Two examples of general n-qubit cycle ($n = 5$ and $n = 6$) are shown in Fig. 2(b).

Note that both X and Y consist of disjoint transpositions and can be executed in parallel using three layers of $\mathbf{CN}$ gates as in Fig. 2(a). As a result, each cycle and the whole permutation can be performed using six layers of $\mathbf{CN}$ gates. This achieves the constant time complexity of a qubit permutation. If auxiliary qubits are used, a cycle can be implemented using only four layers of $\mathbf{CN}$ gates [12].

In addition to permutation, qubit replication (FANOUT) is also an important and nontrivial operation. Qubit replication takes one bit as input and gives two copies of the same bit value as output. If the source qubit is in either $|0\rangle$ or $|1\rangle$, the quantum state can be replicated exactly using a $\mathbf{CN}$ gate. For example, if $|\psi\rangle$ is either $|0\rangle$ or $|1\rangle$, replicating $|\psi\rangle$ to the qubit $|\phi\rangle = |0\rangle$ can be done simply by applying a $\mathbf{CN}$ gate with of $|\psi\rangle$ as the control and $|\phi\rangle$ as the target, i.e., $CN(|\psi, 0\rangle)$. Moreover, since both $|\psi\rangle$ and $|\phi\rangle$ can be used as the source qubits for further replication processes, the number of copies will increase exponentially, which allows C copies of the same quantum state being replicated using only $\log_2 C$ layers of $\mathbf{CN}$ gates.

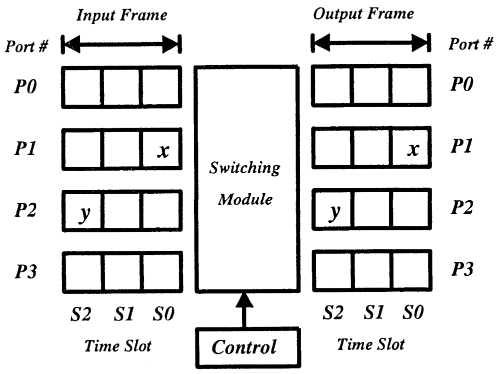


Fig. 3. Typical example of circuit switching.

III. Digital Switching Networks

A. Circuit Switching and Packet Switching

In classical digital communication, switching is needed in order to avoid a fully meshed transmission network. Generally, digital switching technologies fall under two broad categories: *circuit switching* or *packet switching*. In circuit switching, a dedicated path or time slot is reserved for an end-to-end bandwidth demand. The connection is established at the time of call setup and released when the call is torn down. The function of the switching module is to transfer a particular time slot in the input port to a time slot in the output port. For example, in Fig. 3, user x (time slot $S0$ of port $P1$) and user y (time slot $S2$ of port $P2$) are making two-way communication via a 4 × 4 digital switch. For the connection from x to y, the switching module transfers the data from $S0$ of $P1$ to $S2$ of $P2$. Similarly, for the connection from y to x, it transfers the data from $S2$ of$P2$ to $S0$ of $P1$. These operations complete the data exchange between x and y.

Packet switching is more sophisticated than circuit switching. Modern packet switching networks take packets that share the same transmission line as input. A packet can have either a fixed or variable length with a limited maximum size. When a packet arrives at a node, it is stored first and then forwarded to the desired node according to its header. Although significant differences such as *data dependency* and *output contention* exist between circuit switching and packet switching, they still have similarities. In both circuit switching and packet switching, the control subsystem needs to specify the switching configuration for each individual time slot, so the data in that particular time slot can be switched correctly. The switching configuration can be described using a connection digraph, which is introduced in the next section.

B. Connection Digraphs

Before we describe how digital switching can be done using quantum operations, we define a *connection digraph* as follows.

Definition 1: Given an $n \times n$ switch, the connection digraph at time t, $G^t = \{V, E^t\}$, is a digraph such that:

1) each $v_i \in V (i = 0, 1, \ldots n-1)$ represents an I/O port;
2) $\overrightarrow{v_m v_n} \in E^t$ if and only if a connection exists from the input port v_m to the output port v_n at time t.

In a connection digraph, each node represents an I/O port and a directed edge $\overrightarrow{v_m v_n}$ describes an active connection from input port v_m to output port v_n. A digraph G^t describes the connection status of a switch at a specific time, and is called the connection digraph at time t. Note that a directed edge $\overrightarrow{v_m v_n}$ denotes only a one-way data path. For a point-to-point two-way communication between v_m and v_n, both $\overrightarrow{v_m v_n}$ and $\overrightarrow{v_n v_m}$ have to be used. Obviously, due to the connection setup and tear-down processes, the connection digraph is a function of time.

Depending on the status of the switch, the topology of a connection digraph varies. In a general digraph, it is possible that a node has multiple predecessors and multiple successors. However, when there is no output contention or the problem is solved elsewhere, each node will have at most one predecessor. As to the number of successors, it depends on the type of the connection. In a multicast connection, the source node has multiple successors, while in a unicast connection, only a single successor is possible. In this paper, we will discuss the connection digraph based on this model and show that any connection digraph can be built from a set of elementary topologies. These elementary topologies are defined as follows.

Definition 2: Given a digraph $G = (V, E)$ with only one node, i.e., $V = \{v\}$, G is called a *null point* if $E = \emptyset$. Otherwise G is called a *loopback* when $E = \{\overrightarrow{vv}\}$.

In a connection digraph, a null point without predecessor and successor means there is neither input traffic coming from that port nor output traffic going to that port. For a port without incoming traffic, we assume the stuff bits are all 0's. However, a

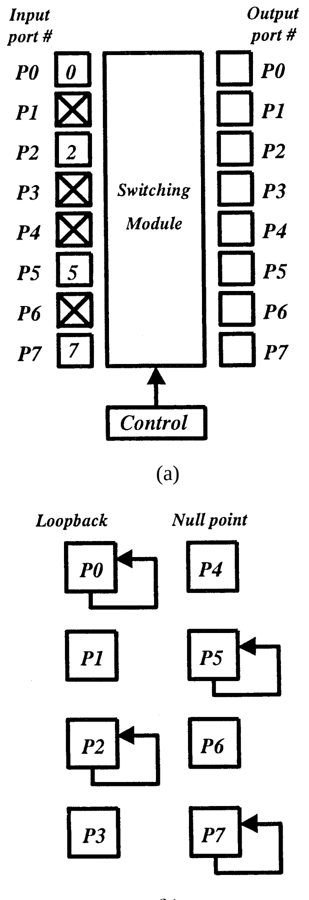


Fig. 4. (a) Connection with null points and loopbacks and (b) its connection digraph.

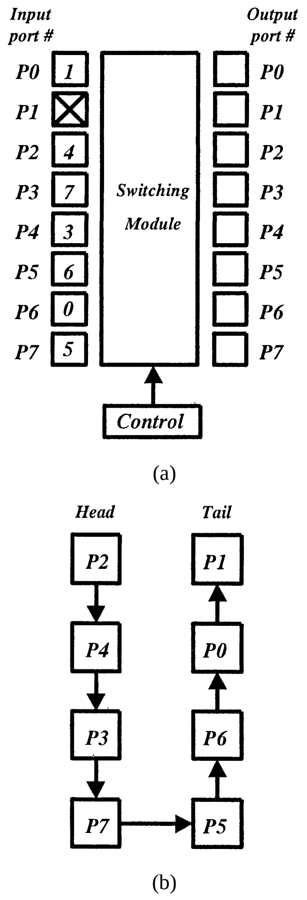


Fig. 5. Example of: (a) a queue connection and (b) its connection digraph.

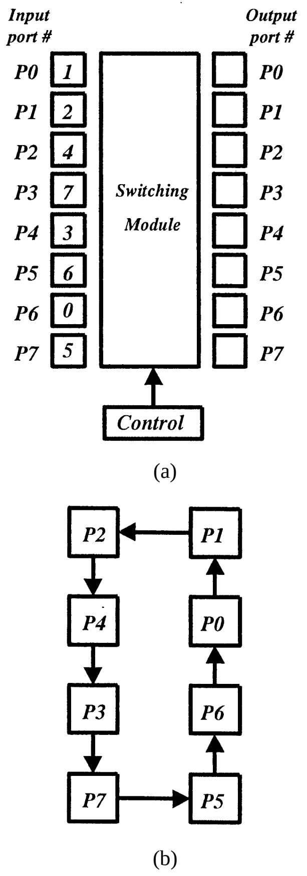


Fig. 6. Example of: (a) a cycle connection and (b) its connection digraph.

single node with a directed edge to itself means the input traffic goes to the same port for output. This trivial cycle effectively denotes a loopback. A loopback G^L can be made from a null point G^N simply by linking the null point to itself. G^L is called the extension loopback of G^N, denoted by $E(G^N)$. An example consists of null points and loopbacks is shown in Fig. 4(a). The numbers in the boxes represents the destination port numbers. An "X" represents no input traffic. Its corresponding connection digraph is depicted in Fig. 4(b).

Definition 3: Given a connected digraph $G = (V, E)$ with $n(n \geq 2)$ nodes, G is called a *queue* if:

1) there exists one and only one *head* $v_h \in V$, such that for each $v_i \in V$, $\overrightarrow{v_i v_h} \notin E$;
2) there exists one and only one *tail* $v_t \in V$, such that for each $v_i \in V$, $\overrightarrow{v_t v_i} \notin E$;
3) for each $v_i \in V (i \neq t)$, there exists one and only one v_j, such that $\overrightarrow{v_i v_j} \in E$.

A queue can be represented as a linear array from the head v_h to the tail v_t, and is denoted as $[v_h, v_1, v_2, \ldots, v_{n-2}, v_t]$. This notation means the connection at a given time includes $\overrightarrow{v_h v_1}, \overrightarrow{v_1 v_2}, \ldots$, and $\overrightarrow{v_{n-2}, v_t}$. Note that there is no input traffic coming from v_t and no output traffic going to v_h. An example of a queue connection is shown in Fig. 5(a), with its connection digraph $G^Q = [P2, P4, P3, P7, P5, P6, P0, P1]$ depicted in Fig. 5(b). Each connection in a queue is apparently a unicast connection, because there is at most one outgoing arrow from each node. Connecting the tail to the head of a queue forms a *cycle*,which is defined in the following definition.

Definition 4: Given a connected digraph $G = (V, E)$ with $n(n \geq 2)$ nodes, G is called a cycle if:

1) for each $v_i \in V$, there exists one and only one v_j, such that $\overrightarrow{v_j v_i} \in E$;
2) for each $v_i \in V$, there exists one and only one v_k, such that $\overrightarrow{v_i v_k} \in E$.

Similarly, a cycle connection can be represented as $(v_0, v_1, v_2, \ldots, v_{n-2}, v_{n-1})$. This means the connection at a given time includes $\overrightarrow{v_0 v_1}, \overrightarrow{v_1 v_2}, \ldots, \overrightarrow{v_{n-2} v_{n-1}}$, and $\overrightarrow{v_{n-1}, v_0}$. In the case of a cycle, each port has its input as well as output. As described earlier, the tail and head of a queue G^Q can be connected to form a cycle G^C. G^C is called the extension cycle of G^Q, denoted by $E(G^Q)$. An example of a cycle connection is shown in Fig. 6(a), with its connection digraph $G^C = (P2, P4, P3, P7, P5, P6, P0, P1)$ depicted in Fig. 6(b).

In order to describe a multicast connection, a *tree* is defined in the following way.

Definition 5: Given a connected digraph $G = (V, E)$ with $n(n \geq 2)$ nodes, G is called a tree if:

1) there exists one and only one *root* $v_r \in V$, such that for each $v_i \in V$, $\overrightarrow{v_i v_r} \notin E$;
2) there exists a collection of nodes L called leaves, such that for each $v_l \in L$ and $v_i \in V$, $\overrightarrow{v_l v_i} \notin E$;
3) for each $v_i \in V - L$, there exists at least one v_j, such that $\overrightarrow{v_i v_j} \in E$.

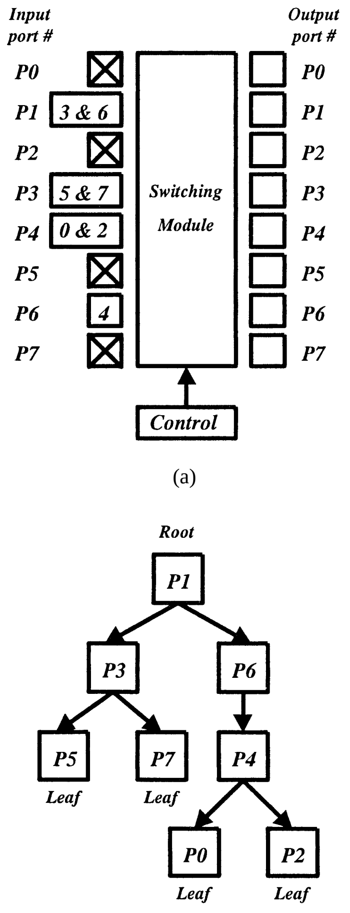


Fig. 7. Example of: (a) a tree connection and (b) its connection digraph.

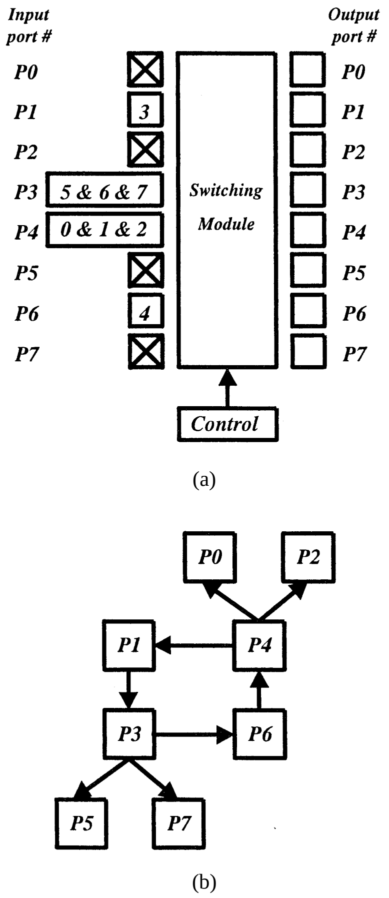


Fig. 8. Example of: (a) a forest connection and (b) its connection digraph.

The nodes in a tree can be divided into three categories: root, internal nodes, and leaves. For the root, the output data is directed to possibly multiple output ports, but no data goes to the root. However, all leaves receive data without generating traffic. All internal nodes have exactly one predecessor and at least one successor. A tree can be represented as a concatenation of queues like $G^T = [v_h^0, \dots v_t^0][v_h^1, \dots v_t^1] \dots [v_h^n, \dots v_t^n]$, with v_h^0 be the root and each of the $v_h^n (n \geq 1)$ be the tail of one of the previous queues. An example of a tree connection is shown in Fig. 7(a). If there are multiple numbers in a box, they represent a multicast connection. Its corresponding connection digraph $G^T = [P1][P1, P3][P1, P6, P4][P3, P5][P3, P7][P4, P0][P4, P2]$ is depicted in Fig. 7(b). Note that a queue is a special case of trees, with each node having only one successor. Connecting any leaf to the root of a tree forms a *forest*, which is defined as follows.

Definition 6: Given a connected digraph$G = (V, E)$ with n nodes $(n \geq 2)$, G is called a forest if:

1) there is one and only one cycle $G^C = (V^C, E^C)$ exists as a sub-digraph of G;
2) let $G' = \{\overrightarrow{v_i v_j} \mid v_i \in V^C, \overrightarrow{v_i v_j} \in E, \overrightarrow{v_i v_j} \notin E^C\}$. $G - G'$ contains the cycle G^C and a collection of disjointed null points, queues, and/or trees;
3) each v_j is either one of the null points, the head of a queue, or the root of a tree in $G - G'$.

A forest basically contains one and only one cycle $G^C = (V^C, E^C)$ as a subdigraph, with some of its nodes linked to either a null point, the head of a queue, or the root of a tree. Following this structure, a forest can be represented by $G^F = \{G^C, G^1, G^2, G^3 \cdots\}$, where $G^1, G^2, G^3, \dots$ be either a null point, a queue, or a tree. A forest can be extended from a tree by connecting any leaf to the root. A forest G_l^F formed by connecting the leave l with the root of G^T is called the extension forest of G^T, denoted by $E_l(G^T)$. An example of a forest connection is shown in Fig. 8(a), with its connection digraph $G^F = \{(P4, P1, P3, P6), [P3][P3, P5], [P3, P7], [P4][P4, P2][P4, P6]\}$ depicted in Fig. 8(b).

Since each node in a unicast connection has at most one successor, a unicast connection digraph only consists of disjoint null points, loopbacks, queues, and/or cycles as subdigraphs. However, a multicast connection switches the data from one node to multiple successors, so a multicast connection digraph consists of disjoint null points, loopbacks, queues, cycles, trees, and/or forests as subdigraphs. Based on these results, we describe the architecture of quantum switching and show how it can be used to implement a connection digraph in the next section.

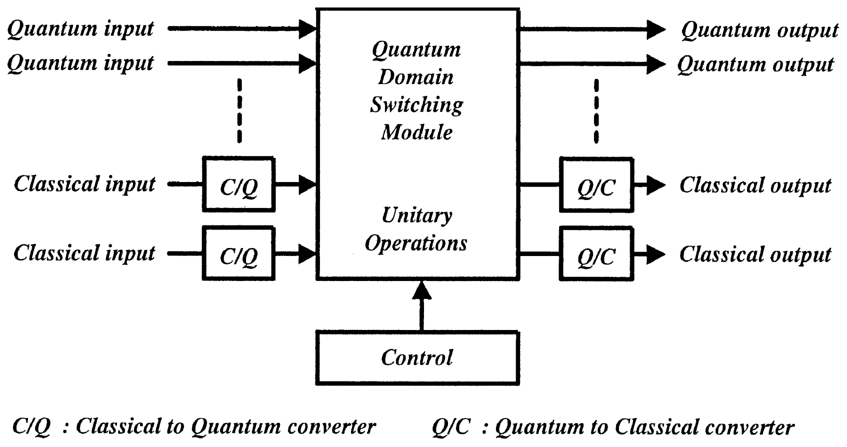


Fig. 9. Architecture of a digital quantum switch.

IV. Digital Quantum Switching

A. Principle of Digital Quantum Switching

The proposed architecture for building a digital *quantum switch* is depicted in Fig. 9. As shown in this figure, the I/O port of a quantum switch can be either quantum- or classical-oriented. For those I/O ports carry quantum information (i.e., qubits), all incoming qubits can be permuted (i.e., switched) using the unitary operations specified by the control subsystem. As will be described later in this paper, this can be done efficiently using $\mathbf{CN}$ gates. In addition to switching qubits, this architecture can also be used to switch classical information. To do so, first we have to convert the classical data into qubits. For example, in a quantum switch with classical I/O ports, a classical to quantum converter (C/Q) is used to convert classical input into qubits. In a C/Q, "0" is converted into $|0\rangle$ and "1" is converted into $|1\rangle$. All qubits are then permuted using unitary operations. After the permutation, all qubits are converted back into their classical form by a quantum to classical converter (Q/C). Note that, if the incoming qubit of a quantum-oriented port is a superposition of $|0\rangle$ and $|1\rangle$, its destination should be a quantum-oriented port so the qubit can be recovered exactly. If a quantum-oriented port is connected to a classical-oriented port, information in the qubit will get lost due to the conversion at Q/C.

B. Connection Digraph Implementation

In this section, we show how a connection digraph can be implemented using $\mathbf{CN}$ gates. First we describe the connection digraph transformation guideline, then we demonstrate how this guideline can be used to implement a connection digraph. Both unicasting and multicasting will be covered in detail.

1) Guidelines for Implementing a Connection Digraph: As described earlier, due to the nature of the connection, unicasting and multicasting have different types of connection digraphs. The digraph of a unicast connection has a collection of disjointed null points, loopbacks, queues, and/or cycles as subdigraphs. However, in the digraph of a multicast connection, subdigraphs such as trees and forests are possible. As a matter of fact, these topologies are inter-related. This is shown in Fig. 10 and summarized as follows:

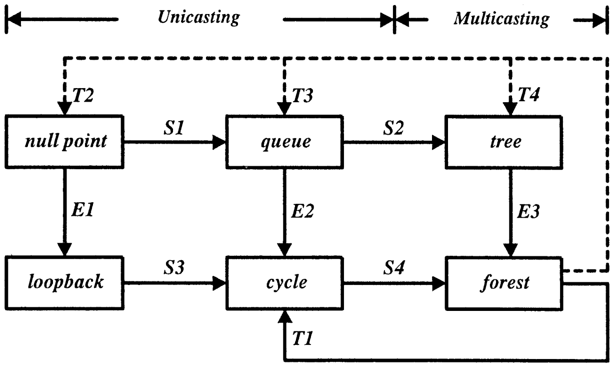


Fig. 10. Interrelated connection topologies.

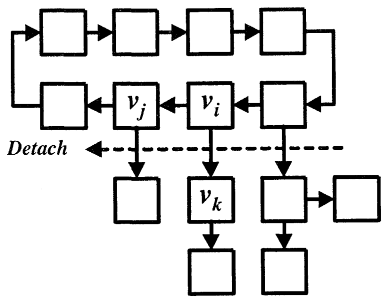
1) a null point can be regarded as a special case of queue, denoted by the arrow S1;
2) a queue can be regarded as a special case of tree, denoted by the arrow S2;
3) a loopback can be regarded as a special case of cycle, denoted by the arrow S3;
4) a cycle can be regarded as a special case of forest, denoted by the arrow S4.

Of course, the binary relation *'is a special case of'* is transitive, so a null point and a loopback are special cases of tree and forest respectively. Fig. 10 also shows the binary relation *'can be extended to'* as follows:

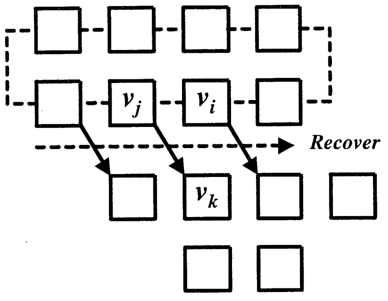
1) a null point G^N can be extended to a loopback $G^L = E(G^N)$, denoted by the process E1;
2) a queue G^Q can be extended to a cycle $G^C = E(G^Q)$, denoted by the process E2;
3) a tree G^T can be extended to a forest $G^F = E_l(G^T)$, denoted by the process E3.

Note that the process of extension only transfers the incoming data from an idle inlet (all 0's) to an outlet which has no outgoing traffic, this does not change the switching function.

The first step of our guideline for implementing a connection digraph is to transform each disjointed subdigraph into loopbacks and/or cycles. Since no circuit is needed to implement a loopback and only six layers of $\mathbf{CN}$ gates are sufficient



(a)



(b)

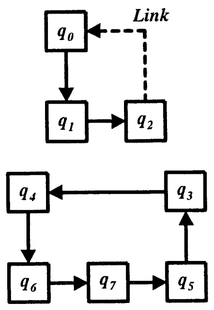
Fig. 11. Example of: (a) cycle extraction and (b) link recovery.

to implement a cycle, the switching process can be done efficiently. Some of these transformations are straightforward. For example, following E1, a null point G^N can be extended to a loopback $G^L = E(G^N)$. Also, following E2, a queue G^Q can be extended to a cycle $G^C = E(G^Q)$. However, for a tree or a forest, *'cycle extraction'* and *'link recovery'* have to be used. The process of cycle extraction and link recovery are described as follows.

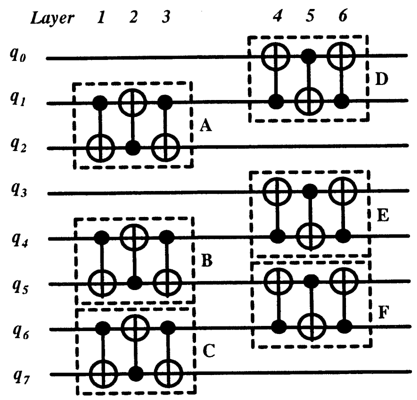
Cycle Extraction: By definition, a forest contains one and only one cycle $G^C = (V^C, E^C)$ with a subset of V^C linked to a null point, the head of a queue, or the root of a tree. The process of cycle extraction detaches all the null points, queues, and trees from the cycle by cutting all the edges in$E = \{\overrightarrow{v_iv_j} \mid v_i \in V^C, v_j \notin V^C\}$, as shown in Fig. 11(a). This procedure transforms a forest into one cycle (arrow T1 in Fig. 10) and a collection of null points, queues, and/or trees (arrow T2, T3, and T4, respectively). Each of the null points and queues can further be transformed into loopbacks and cycles via process E1 and E2. If there are still any trees in the remaining digraph, extensions can be made again to transform the trees into forests (process E3) and the procedure of cycle extraction can be applied recursively (arrow T1, T2, T3, and T4) until no trees are left. This procedure eventually transforms a forest into loopbacks and/or cycles, so that the permutation can be implemented using six layers of **CN** gates in parallel.

Link Recovery: After each cycle has been implemented, the links that had been cut must be recovered. That is, for each $\overrightarrow{v_iv_j} \in E^C$, if $\overrightarrow{v_iv_k} \in E$ but $\overrightarrow{v_iv_k} \notin E^C$, v_j must be replicated to v_k, as shown in Fig. 11(b). Since there will be at most $n-2$ such k's in a multicast connection digraph, in the worst case the replication can be done by $\log_2 n$ layers of **CN** gates.

After implementing cycle extraction and link recovery, the reduction process of a forest is completed. For a tree, it can be extended first to a forest via process E3 and then follows the same algorithm to complete the reduction. In summary, all the elementary topologies can be reduced to a collection of loopbacks and cycles. This allows an efficient implementation of the switching process.



(a)



(b)

Fig. 12. (a) Unicast connection digraph and (b) its quantum circuits.

2) Unicast Quantum Switching: Following the guideline described above, in this section we show how a unicast connection digraph can be implemented with a time complexity of $O(1)$ and a space complexity of $O(n)$. A typical unicast connection status at a given time is shown by the solid arrows in Fig. 12(a). In this example, two connection subdigraphs need to be implemented

$$G^C = (q_3, q_4, q_6, q_7, q_5) \quad (25)$$
$$G^Q = [q_0, q_1, q_2]. \quad (26)$$

These can be done by first extending G^Q to $G^{C'} = (q_0, q_1, q_2)$, as shown by the dash link in Fig. 12(a), and then implementing G^C and $G^{C'}$ using six layers of **CN** gates. As described previously, the subdigraph $G^C = (q_3, q_4, q_6, q_7, q_5)$ can be done by first applying

$$X = (q_6, q_7)(q_4, q_5) \quad (27)$$

and then

$$Y = (q_6, q_5)(q_4, q_3). \quad (28)$$

The transposition (q_4, q_5) is done with

$$(q_4, q_5) = CN(q_4, q_5) \cdot CN(q_5, q_4) \cdot CN(q_4, q_5) \quad (29)$$

as shown by block B in Fig. 12(b). In the same way, (q_6, q_7), (q_4, q_3), (q_6, q_5) are done by blocks C, E, and F, respectively.

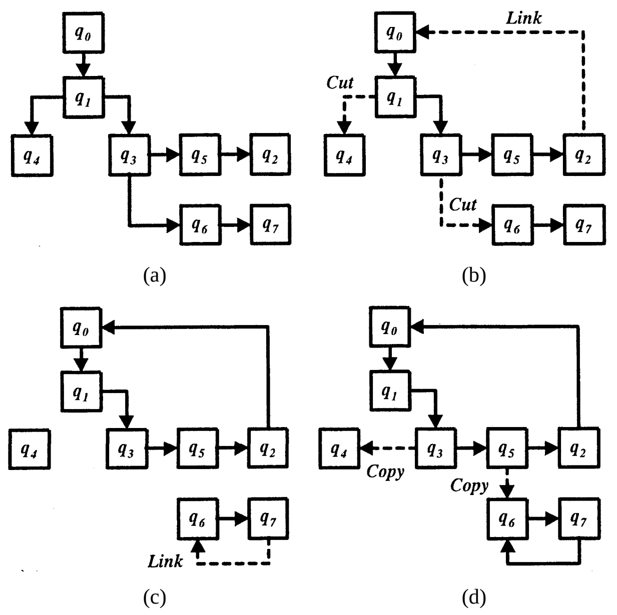


Fig. 13. Procedures for implementing a multicast connection digraph.

Similarly, the implementation of $G^{C'} = (q_0, q_1, q_2)$ can be done by first applying $X = (q_1, q_2)$ and then $Y = (q_1, q_0)$. These are implemented as blocks A and D in Fig. 12(b).

Note that, independent of the switch size n, the whole circuit can be completed in six layers of **CN** gates over n qubits. This achieves a time complexity of $O(1)$ and a space complexity of $O(n)$.

3) Multicast Quantum Switching: In classical packet switching, the input packets are usually buffered in the memory, multicasting can be easily achieved by reading the packet once and writing the same packet to multiple destinations. If the switching is performed using quantum operations, multicasting can be done by replicating the input qubit to multiple destination qubits. A typical multicasting configuration is shown in Fig. 13(a).

In this example, the following connection digraph needs to be implemented:

$$G^T = [q_0, q_1][q_1, q_4][q_1, q_3][q_3, q_5, q_2][q_3, q_6, q_7]. \tag{30}$$

Based on the guidelines, each of the steps is shown below.

1) The tree G^T can be extended to a forest by linking any leaf, say q_2, to q_0. The cycle extraction procedure is then performed to cut $\overrightarrow{q_1q_4}$ and $\overrightarrow{q_3q_6}$. The result is shown in Fig. 13(b).
2) The extension and cycle extraction processes are recursively applied to $[q_6, q_7]$ until no tree is left, as shown in Fig. 13(c).
3) Each of the disjointed subdigraphs can be implemented in parallel. The subdigraph $G^C = (q_0, q_1, q_3, q_5, q_2)$ can be done by first applying $X = (q_1, q_2)(q_3, q_5)$ and then $Y = (q_1, q_0)(q_3, q_2)$, while $G^{C'} = (q_6, q_7)$ can be implemented directly, as shown by blocks A, B, D, E, and C in Fig. 14.
4) Each of the disconnected edges has to be recovered, so q_3 needs to be replicated to q_4, and q_5 needs to be replicated to q_6, as shown in Fig. 13(d). These can be done by blocks F and G in Fig. 14.

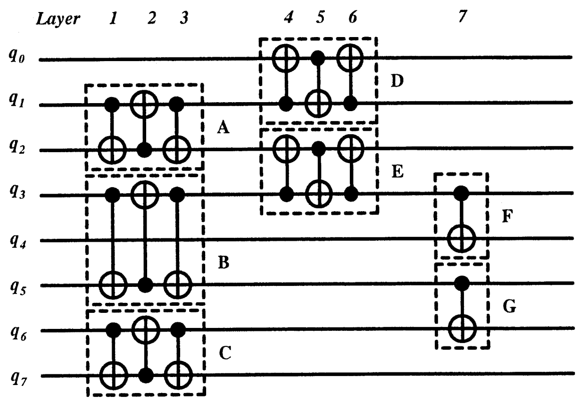


Fig. 14. Quantum circuits for a multicast connection digraph.

In general, the total number of layers for implementing a multicast connection digraph is $6 + \lceil \log_2(r+1) \rceil$, where r is the maximum number of $\overrightarrow{v_jv_k}(k = 1, 2, \ldots r)$ that are to be recovered. In the worst case, when one inlet is broadcast to all other $n-1$ outlets, the whole connection digraph can be done in $O(\log_2 n)$ layers of **CN** gates over n qubits. This results in a time complexity of $O(\log_2 n)$ and a space complexity of $O(n)$.

C. Issues on Physical Implementation

Several physical implementations have been demonstrated to be good candidates for performing quantum operations [13]–[17]. Some issues on selecting the qubit for physical implementation are discussed in this section.

First, just like any other quantum system, decoherence and error correction are important issues in such a quantum switch. In a general quantum system, the qubit selection needs to maximize

$$S_{\max} = \frac{t_o}{t_d} \tag{31}$$

where t_d is the decoherence time and t_o is the time for a single quantum operation. This allows more quantum operations to be completed within the decoherence time. However, because there are only six layers of **CN** gates to be performed before the qubits lose their coherence, maximizing $S_{\max}$ is not such an important issue. Moreover, it turns out that selecting an implementation with minimum t_o will be useful. To be more specific, as long as $6 * t_o \leq t_d$, the line speed of the switch can be as high as $1/(6 * t_o)$ b/s. For some implementations, this is on the order of 10^{-14} to 10^{-19} second [18]. To further reduce the probability of errors, many error correction schemes can be used. For example, a qubit that carries one bit of information can be encoded using m qubits [19]–[21]. This protects the qubits from a limited number of errors. Fortunately, since the operation time $6 * t_o$ is relatively short compared with the decoherence time of most implementations, errors tend to be small enough such that a small$m$ is suffice to do the protection.

Second, if this architecture is used to switch classical information, the C/Q and Q/C form the interface between the classical domain and the quantum domain. Due to wide applications of optical devices in the current telecommunication industry, we assume the incoming classical data is in optical form (although the frequency and intensity are not designed specifically for classical-to-quantum conversion). The implementation of C/Q depends on what kind of qubit is used in the switching module. For example, if the energy level of an electron is used as the qubit, the incoming optical data "1" must be able to excite the energy level of the electron from $|0\rangle$ to $|1\rangle$.On the other hand, the Q/C should be able to convert the quantum state $|0\rangle$ and $|1\rangle$ back to their optical form. Usually this can be done by performing a measurement on the qubit.

D. Advantages of Quantum Switching

In the field of classical digital switching, various techniques have been used to switch the input data to the corresponding output port. For example, data can be switched in the space domain, the time domain, etc. If the data is switched in the space domain, usually the switching process is performed in parallel and can be completed in constant, i.e., $O(1)$, time. However, to provide this capability, usually the space complexity is a tradeoff. For example, in the crossbar architecture, it requires an $O(n^2)$ space consumption to perform the switching. On the other hand, if the data is switched in the time domain, it needs only $O(n)$ space complexity in general. However, in a time division switch, connections are established in a time-sharing manner so a connection occupies the resources for only a short duration of time. Although switching in the time domain cannot be "blocked," the time duration for switching each connection eventually will be limited by the devices (e.g., memory) due to the increase of total throughput. In other words, although the space consumption is usually $O(n)$ (which is reasonable), time/speed is the bottleneck for a time switch.

Based on these discussions, we show that quantum switching has a lot of advantages over classical switching technologies. The first advantage of performing digital switching using operations in the quantum domain is that it is *strict-sense nonblocking*. A switch is called strict-sense nonblocking if the network can always connect each idle inlet to an arbitrary idle outlet independent of the current network permutation [22]. Note that switching in the space domain is not always nonblocking. Sometimes, the required data path can not be established even if the output port is available. However, switching in the quantum domain is actually a unitary transformation, which is always possible. This results in the fact that a quantum switch is nonblocking in the strict sense.

Second, although it has the same time complexity $(O(1))$ as a space switch, quantum switching achieves a better performance in terms of the space complexity. To make a classical space switch nonblocking, a certain number of modules in the middle stage have to be used to allocate a physical path for each connection, so the number of cross-points increases with the size of the switch. For example, in a Close network [23], the minimum number of modules in the middle stage is $2n-1$ and it requires at least $4n(\sqrt{2n}-1)$ cross-points to build the network. However, an $n \times n$ quantum switch uses only $O(n)$ qubits as the basis to perform the switching while it keeps the same time complexity as a space switch.

Third, although it has the same space complexity $(O(n))$ as a time switch, quantum switching achieves a better performance in terms of the time complexity. In a classical time switch, usually the bottleneck is the speed of the switching device. Because when the throughput increases, the time duration for switching a particular bit of data decreases. For example, in a memory switch with throughput T, the memory speed must be at least $1/(2T)$ to allow one read and one write operation to be performed. However, in the quantum switching, the time complexity is scalable and not sensitive to the throughput. A high throughput quantum switch can be built simply by increasing the number of I/O ports. This induces only a reasonable amount $(O(n))$ of space consumption. Even in the worst case scenario, the throughput gain still outweights the time penalty in a classical time domain switch ($O(n)$ versus $O(\log_2 n)$).

V. Conclusions

Networks are growing rapidly due to increased number of users and rising demands for bandwidth-intensive services. To support such a huge traffic volume, a wide range of different technologies are being proposed as the core of a high-performance switch. In this paper, an architecture of digital quantum switching is presented. The proposed mechanism allows digital data to be switched using a series of quantum operations. The procedures of how to implement unicast and multicast connections are discussed in detail. In terms of the blocking rate, this architecture is strict-sense nonblocking. From a complexity point of view, the space complexity grows only linearly with the number of I/O ports, and the time complexity is constant for unicasting and logarithmic for multicasting. This architecture is scalable and can be applied to build high-performance switching devices.

References

[1] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *J. Stat. Phys.*, vol. 22, no. 5, pp. 563–591, 1980.

[2] R. Feynman, "Simulating physics with computers," *Int. J. Theor.Phys.*, vol. 21, pp. 467–488, 1982.

[3] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," in *Proc. R. Soc. Lond. A*, vol. 400, 1985, pp. 97–117.

[4] C. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proc. IEEE Int. Conf. Computers Systems and Signal Processing*, 1984, pp. 175–179.

[5] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc.. 35th Annu. IEEE Symp. Foundations of Computer Science*, 1994, pp. 124–134.

[6] L. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory of Computing*, 1996, pp. 212–219.

[7] R. Jozsa, D. Abrams, J. Dowling, and C. Williams. (2000, Apr.) Quantum clock synchronization based on shared prior entanglement. [Online] Available: http://www.arXive.org/quant-ph/0004105/

[8] I. Chuang. (2000, May) Quantum algorithm for distributed clock synchronization. [Online] Available: http://www.arXive.org/quant-ph/0005092/

[9] I. M. Tsai and S. Y. Kuo, "Quantum boolean circuit construction and layout under locality constraint," in *Proc. 1st IEEE Conf. Nanotechnology*, 2001, pp. 111–116.

[10] D. DiVincenzo, "Two-bit gates are universal for quantum computation," *Phys. Rev. A*, vol. 51, no. 2, pp. 1015–1022, 1995.

[11] A. Barenco, "A universal two bit gate for quantum computation," in *Proc. R. Soc. Lond. A*, vol. 449, 1995, pp. 679–683.

[12] C. Moore and M. Nilsson. (1998, Aug.) Parallel quantum computation and quantum codes. [Online] Avaialble: http://www.arXive.org/quantph/9808027/

[13] N. Gershenfeld and I. Chuang, "Bulk spin resonance quantum computation," *Science*, vol. 275, pp. 350–356, 1997.

[14] J. Cirac and P. Zoller, "Quantum computation with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, 1995.

[15] Q. Turchette, C. Hood, W. Lange, H. Mabuchi, and H. Kimble, "Measurement of conditional phase shifts for quantum logic," *Phys. Rev. Lett.*, vol. 75, pp. 4710–4713, 1995.

[16] D. Loss and D. DiVincenzo, "Quantum computation with quantum dots," *Phys. Rev. A*, vol. 57, pp. 120–126, 1998.

[17] B. Kane, "A silicon-based nuclear spin quantum computer," *Nature*, vol. 393, pp. 133–137, 1998.

[18] D. DiVincenzo, "Quantum computation," *Science*, vol. 270, pp. 255–261, 1995.

[19] S. Lloyd, "A potentially realizable quantum computer," *Science*, vol. 261, pp. 1569–1571, 1993.

[20] A. Berthiaume, D. Deutsch, and R. Josza, "The stabilization of quantum computations," in *Proc. 4th Workshop on Physics and Computation*, 1994, pp. 60–62.

[21] P. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, vol. 52, pp. 2493–2496, 1995.

[22] A. Pattavina, *Switching Theory*. West Sussex, U.K.: Wiley, 1998, p. 54.

[23] C. Clos, "A study of nonblocking switching networks," *Bell Syst. Tech. J.*, vol. 32, no. 2, pp. 406–424, 1953.

I-Ming Tsai was born in Taipei, Taiwan, R.O.C., in 1966. He received the B.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 1988, and the M.S. degree in telecommunications from the University of Pittsburgh, Pittsburgh, PA, in 1993. He is currently working toward the Ph.D. degree at National Taiwan University, Taipei, Taiwan, R.O.C.

Since 1993, he has been with Chunghwa Telecommunication Laboratories, Taoyuan, Taiwan, R.O.C., where he works on broadband switching, network design, and network architecture evolution. His research fields include experimental and theoretical aspects of quantum computing, quantum communication, and nanotechnology.

Sy-Yen Kuo (S'85–M'88–SM'98–F'01) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1979, the M.S. degree in electrical and computer engineering from the University of California, Santa Barbara, in 1982, and the Ph.D. degree in computer science from the University of Illinois, Urbana-Champaign, in 1987.

Since 1991, he has been with National Taiwan University, where he is currently a Professor and the Chairman of Department of Electrical Engineering. He spent his sabbatical year as a Visiting Researcher at AT&T Labs-Research, NJ, from 1999 to 2000. He was the Chairman of the Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan, R.O.C., from 1995 to 1998, a faculty member in the Department of Electrical and Computer Engineering, University of Arizona, from 1988 to 1991, and an Engineer with Fairchild Semiconductor and Silvar-Lisco, both in California, from 1982 to 1984. In 1989, he also was a Summer Faculty Fellow in the Jet Propulsion Laboratory, California Institute of Technology. His current research interests include software reliability engineering, quantum computing, dependable systems and networks, and optical WDM networks. He has authored more than 180 papers published in journals and conference proceedings.

Prof. Kuo received the Distinguished Research Award (1997–2004) from the National Science Council, Taiwan, R.O.C. He was also a recipient of the Best Paper Award at the 1996 International Symposium on Software Reliability Engineering, the Best Paper Award in the simulation and test category at the 1986 IEEE/ACM Design Automation Conference (DAC), the National Science Foundation's Research Initiation Award in 1989, and the IEEE/ACM Design Automation Scholarship in 1990 and 1991.