Efficient Adder Circuits Based on a Conservative Reversible Logic Gate

J.W. Bruce, M.A. Thornton, L. Shivakumaraiah, P.S. Kokate, and X. Li

Department of Electrical and Computer Engineering
Mississippi State University, Mississippi State, MS 39762-9571 USA
{jwbruce,mitch}@ece.msstate.edu

Abstract

Conservative and reversible logic gates are widely known to be compatible with revolutionary computing paradigms such as optical and quantum computing. A fundamental conservative reversible logic gate is the Fredkin gate. This paper presents efficient adder circuits based on the Fredkin gate. Novel full adder circuits using Fredkin gates are proposed which have lower hardware complexity than the current state-of-the-art, while generating the additional signals required for carry skip adder architectures. The traditional ripple carry adder and several carry skip adder topologies are compared. Theoretical performance of each adder is determined and compared. Although the variable sized block carry skip adder is determined to have shorter delay than the fixed block size carry skip adder, the performance gains are not sufficient to warrant the required additional hardware complexity.

1. Introduction

As proved by Landauer [4], using traditional irreversible logic gates such as AND or multiplexer types inevitably leads to energy dissipation in a circuit regardless of the realization technology. Bennett [2] showed that for power not to be dissipated in an arbitrary circuit it is necessary that the circuit be built from reversible gates. Reversible logic gates are circuits that have the same number of inputs and outputs and have one-to-one and onto mappings between vectors of inputs and outputs; thus, the vector of input states can be always reconstructed from the vector of output states. Of course, Bennett's theorem is only a necessary and not sufficient condition, but its extreme importance is in the fact that it points out that every future technology will have to use reversible gates to reduce power when it will be impossible for classical technologies to keep Moore's Law functioning.

Optical logic gates have been proposed to perform classic (irreversible) logic functions. Optical logic gates have the potential to work at macroscopic (light pulses carry information), or quantum (single photons carry information) levels with great efficiency. Although a few functional optical (classic) logic gate designs exist, producing all logic functions is difficult [10]. The potential of optical computing cannot be realized without logic functions compatible with optical systems. Reversible and conservative logic functions are natural for optical [6].

Unlike classical bits that are realized as electrical voltages or currents present on a wire, quantum logic operations manipulate qubits [1]. Qubits are microscopic entities such as a photon or atomic spin. Boolean quantities of 0 and 1 are represented by a pair of distinguishable different states of a qubit. Because the qubit probabilities must be preserved at the output of the quantum gate, it is noted that all matrices representing them are unitary. An important unitary matrix property is that of full rank. This property implies that quantum gate matrix rows and columns are orthonormal. Therefore, past results from spectral methods for classic digital logic [12] are directly applicable to quantum logic synthesis. Furthermore, since quantum logic gates are represented by unitary orthonormal matrices, they represent logically reversible gates.

Adders are fundamental building blocks in many computational units. The anticipated paradigm shift to logic compatible with optical and quantum computing requires compatible adder implementations. Faster adder (generate and propagate carries with minimum delay) circuits have been investigated for several decades. This paper continues that tradition by describing several adders using the Fredkin gate (FG), a conservative reversible logic gate compatible with optical and quantum computing. Section 2 provides definitions and background on reversible and conservative logic concepts. This section also introduces classical (irreversible) binary logic gates implemented using FGs. Section 3 describes several ripple carry and carry skip adders implemented in classical binary logic and FGs. Also, Section 3 describes some optimizations for conservative reversible logic gate adders. Section 3 also compares size, performance, and implementation specific issues concerning the proposed adder

circuits. Section 4 gives some preliminary conclusions and ideas for future circuit refinements.

2. Principles of conservative reversible logic

A gate is reversible if the gate's inputs and outputs have a one-to-one correspondence, i.e. there is a distinct output assignment for each distinct input. Therefore, a reversible gate's inputs can be uniquely determined from its outputs. A ramification of this definition is that reversible logic gates must have an equal number of inputs and outputs.

If logic gates are classified according to the number of inputs and outputs, a gate with m inputs and n outputs is a (m,n) logic gate, where reversible gates must have $m = n$. Furthermore, a (n,n) reversible gate's output vector is a permutation of the numbers 0 to $2^n - 1$.

A gate is balanced if its output equals one for exactly one-half of its inputs. If a gate is not balanced, it is said to be unbalanced. Reversible gates are balanced. The only nontrivial reversible logic gate in classical digital logic is the (1,1) reversible gate, i.e. the inverter. A gate not possessing the reversibility characteristic is said to be irreversible. All multiple-input single output logic gates in classical digital logic are irreversible, e.g. AND, OR, XOR, etc.

A circuit without constants on inputs which includes only reversible gates realizes on all outputs only balanced functions, therefore it can realize non-balanced functions only with garbage outputs. An additional constraint of reversible logic is that the fanout of every signal, including primary inputs, must be one. In the classical paper [3], Fredkin and Toffoli formulate the synthesis problem for reversible logic. Classical logic synthesis methods cannot be directly applied to design reversible logic circuits, and reversible logic specific synthesis methods do not yet exist [9]. Currently, logic functions constructed with reversible logic gates are designed in an ad hoc fashion.

By definition, all reversible gates have an inverse, i.e. a gate that "undoes" the logic function. A logic gate is said to be self-invertible if the gate is equal to its own inverse. In some works, the terms symmetric or dual are used, and the ambiguous term invertible is also common. For example, a gate G is self-invertible if, for every input x, $G(G(x)) = x$, or, equivalently, $G^2(x) = x$. Obviously, the classic digital inverter is self-invertible.

A gate is zero-preserving if the all zeros input produces all zeros outputs. A gate is one-preserving if the all ones input produces all ones outputs. A gate is said to be controlled if one or more of its inputs causes a function to be performed, conditionally, on its other inputs. The inputs that determine whether the action is taken or not are called the gate's control signals.

A gate is conservative if the Hamming weight (number of logical ones) of its input equals the Hamming weight of its output. A conservative reversible gate is a gate that is both conservative and reversible simultaneously. A consequence of a gate's reversibility and conservability is that conservative reversible gates are zero-preserving and ones-preserving. A conservative reversible logic gate effectively permutes its inputs to form its outputs. Conservative gates are not necessarily reversible as several inputs with equal Hamming weights can be mapped to a single output of the same Hamming weight. For example, consider the two-input, two-output logic gate where one output is the logical AND of the inputs, and the other output is the logical OR of the inputs. This logic gate is conservative and irreversible. Magnetic bubble circuits initiated much research on conservative, but irreversible, logic in the 1970s [5][11].

2.1. The Fredkin gate

Reversible logic elements, both with and without the property of conservatism, can be defined in many different logic systems. For the purposes of this paper, the class of reversible elements that can be modeled as binary-valued logic circuits will be called classical reversible logic elements. Much work has been done in defining and characterizing classical reversible logic [2][3][11], and applying the concepts toward power conservation and specific technology implementations [4][5][6]. This section will briefly define basic results in classical reversible logic.

Consider the (3,3) conservative reversible gate with inputs $X2$, $X1$, $X0$, and outputs $Y2$, $Y1$, $Y0$, where

$$Y0 = \begin{cases} X0, & X2 = 0 \\ X1, & X2 = 1 \end{cases}, \quad Y1 = \begin{cases} X1, & X2 = 0 \\ X0, & X2 = 1 \end{cases},$$

and

$$Y2 = X2.$$

This gate is known as the Fredkin gate (FG) [3]. Fig. 1 shows the FG circuit symbol. The FG is sometimes called the controlled permutation gate. The FG was considered a primitive logical element in magnetic bubble circuits [5].

Over the years, several other reversible, nonconservative logic gates have been proposed, including the Feynman gate, Toffoli gate, Kerntopf gate, Peres gate, Margolus gate, and several gates by De Vos [9].

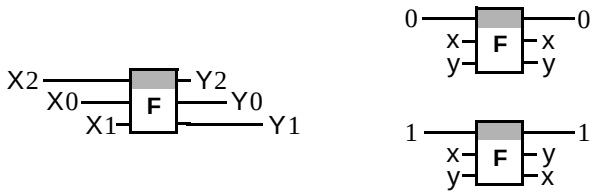

Fig. 1. Fredkin gate circuit symbol and operation

2.2. Classical digital logic using Fredkin gates

The FG can be used to create the inverse and signal duplication (fan-out) functions. Fig. 2 shows the inverse and fan-out function using a FG. The inverse function is formed by tying the FG terminals to the power supplies. Since fan-out is the dual of inverse, a fan-out function is also created.

The two input AND gate (AND2) can be generated by the FG by grounding one terminal. The two input OR gate (OR2) can be generated by the FG by tying one terminal to V_{DD}. Fig. 3 shows the AND2 and OR2 functions using a FG. Obviously, the AND2 and OR2 circuits are irreversible, as are their respective logic functions. Higher order AND and OR gates can be constructed using FGs arranged in a binary tree. A B bit gate requires $B-1$ FGs. An input passes through a maximum of

$$\lceil \log_2 N \rceil \quad (1)$$

FGs.

The two input XOR gate (XOR2) can be generated by connecting two FGs as shown in Fig. 4. Furthermore, the equivalence operation is performed simultaneously. Wider XOR gates can be created by appending an additional FG per input to the circuit in Fig. 4.

3. Adder Circuits

The simplest and most straightforward adder architecture is the ripple carry adder. Ripple carry adders utilize only one major circuit element, the full adder. Each full adder generates the sum and carry bits due to its input addend bits and its carry input bit. Unfortunately, the ripple carry adder delay increases linearly with the width of the adder.

Many other adder circuits have been proposed to reduce the delay in addition [7]. Most circuits strive to compute the more significant carries more quickly. Theoretically, carry lookahead adders represent the best possible performance. However, carry lookahead adders achieve their speed through parallel carry computations, thus, employ a large number of gates. The carry skip adder presents a hardware and performance compromise between the ripple carry adder and the carry lookahead adder. The paper will examine ripple carry and carry skip adder circuits using the simplest conservative reversible logic gate, the FG.

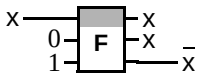

Fig. 2. Fredkin gate implementation of NOT/FAN-OUT

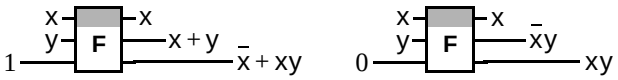

Fig. 3. Fredkin gate implementation of OR2 and AND2

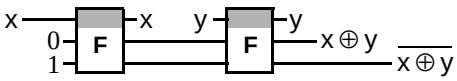

Fig. 4. Fredkin gate implementation of XOR2

3.1. Ripple carry adder

The full adder is the basic building block in the ripple carry adder, and most other adder circuits. The full adder computes the sum bit s_i and carry output bit c_{i+1} based on its addend input bits x_i and y_i, and its carry input bit c_i. The sum bit and carry output bits are

$$s_i = x_i y_i c_i + x_i \bar{y}_i \bar{c}_i + \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i = x_i \oplus y_i \oplus c_i, \quad (2)$$

and

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i. \quad (3)$$

The worst case through the ripple carry adder is when a carry generated in the first stage must "ripple" through all adder stages.

Several researchers have proposed full adder circuits based on the FG. Fig. 5 shows a recently proposed example [8]. Although the circuit in Fig. 5 can be redrawn to avoid the fan-out of y_i, this paper proposes the full adder circuit in Fig. 6. The proposed circuit implements the full adder with one fewer FG and avoids signal fan-out, which is forbidden in a strict reversible computing paradigm.

3.2. Carry skip adder

The carry skip adder reduces the delay due to the carry computation. Consider the full adder's operation. If either input is a logical one, the cell will propagate the carry input to the carry output. Therefore, the ith full adder carry input, c_i, will propagate to its carry output,

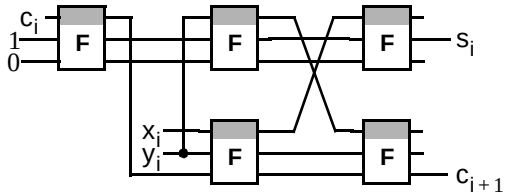

Fig. 5. Fredkin full adder proposed in [8]

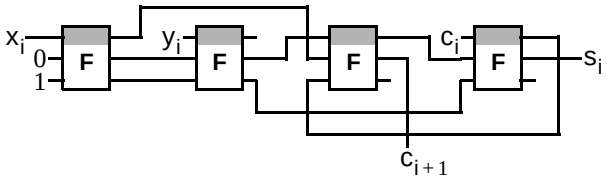

Fig. 6. Fredkin full adder circuit avoiding fan-out

c_{i+1}, when $p_i = x_i \oplus y_i$. Furthermore, multiple full adders, called a block, can generate a "block" propagate signal to detour the incoming carry around to the block's carry output signal. Fig. 7 shows a four bit carry skip adder block. Each block is a small ripple carry adder producing the block's sum and carry bits. However, each block quickly calculates whether the block's carry input is propagated to its carry output.

The block carry input C_{in} is propagated as the block carry output C_{out}, if the block propagate P is one. The block propagate signal is generated with an AND gate. Fig. 8 shows the proposed carry skip compatible full adder constructed with FGs. Strictly reversible logic implementations do not allow fan-out, thus the full adders circuits in Fig. 5 and Fig. 8 are not applicable. Fig. 9 shows a circuit without fan-out created from the addition of a fan-out gate to the circuit in Fig. 8.

The carry skip adder worst case delay is the when the carry generated in the very first full adder ripples through all full adder stages in the first block, generates C_{out} for the first block, skips all the intermediate blocks, and ripples through the full adder stages of the last block.

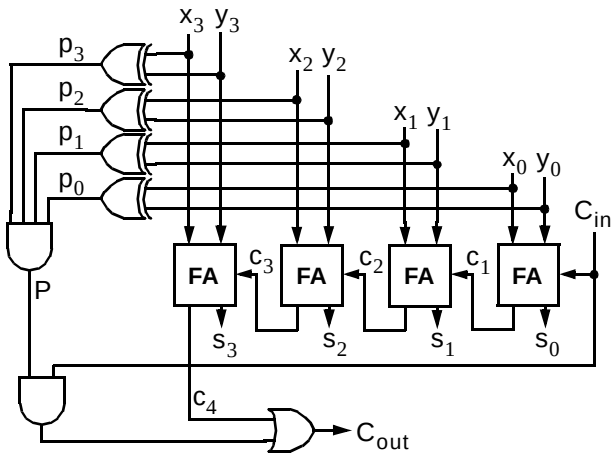

Fig. 7. Four bit carry skip adder block

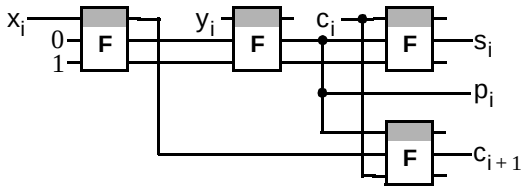

Fig. 8. Fredkin full adder circuit (with propagate)

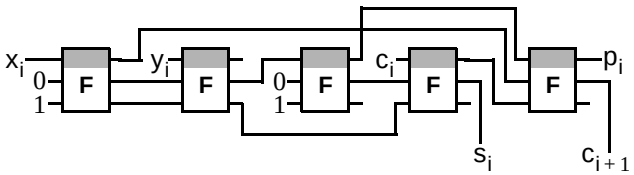

Fig. 9. Fredkin full adder circuit avoiding fan-out

3.3. Carry skip adder block using Fredkin gates

The carry skip adder propagate the block carry input to the next block if block group propagate signal P is one. The conventional skip block in Fig. 7 uses the AND-OR gate combination. A carry skip block based on FGs can be constructed by replacing the AND and OR gates with their respective FG implementation in Fig. 3. However, the carry skip block can be modified to take advantage of the more complex logical function performed by the FG.

The proposed carry skip block replaces the AND2-OR2 gate combination with a single FG. Fig. 10 shows the block diagram of the carry skip adder block constructed with FGs, where the "FFA" block is a Fredkin full adder circuit from either Fig. 8 or Fig. 9. The three FGs in the middle of Fig. 10 perform the AND4 operation generating the block propagate signal P. (The schematic looks awkward. However, recall that FGs are reversible, thus symmetric with respect to inputs and outputs.) The single FG in the left side of Fig. 10 performs the AND-OR function to create the carry skip logic and block carry out signal C_{out}. The FG propagates the block's carry input to the next block if the block propagate signal P is one; otherwise, the FG propagates the most significant full adder carry c_4 to the next block.

The traditional carry skip AND-OR logic in Fig. 7 and the FG carry skip logic in Fig. 10 do not possess equivalent truth tables. The AND-OR carry skip generates a block carry out, i.e. $C_{out} = 1$, when the most significant full adder carry c_4 equals one, the block propagate P equal one, and the block carry input signal C_{in} equals zero. The FG carry skip logic generates block carry out signal $C_{out} = 0$ under the same inputs. Either result is appropriate, because c_4 cannot equal one when $C_{in} = 0$

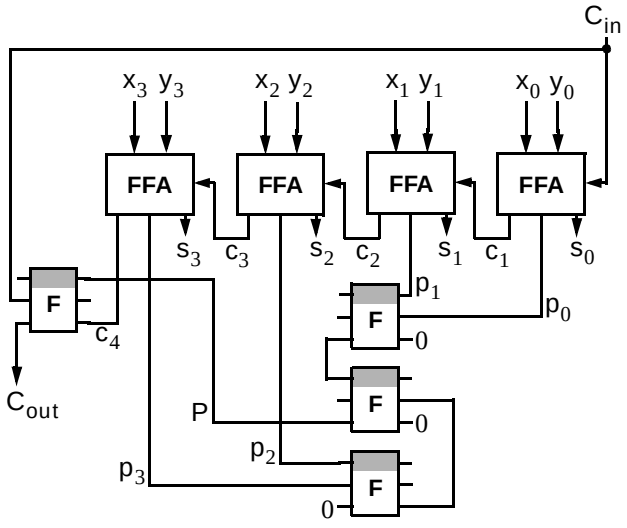

Fig. 10. Four bit Fredkin carry skip adder block

and $P = 1$. However, it must be noted that the Fredkin carry skip logic more faithfully adheres to the spirit of carry skip addition by propagating the correct value of C_{in} to C_{out}. Thereby, carry propagation performance is improved in block inputs that are possible.

The FG carry skip logic in Fig. 10 can improve carry propagation if the block carry propagate signal P is one. When $P = 1$, the block carry input C_{in} should propagate to the next block, regardless of the result of the carry c_4 created by the full adders within the block. Because the traditional AND-OR carry skip logic in Fig. 7 must account for sending c_4 when $P = 0$, it does not perform its carry skip operation efficiently. Consider the carry skip adder block in Fig. 7 when $C_{in} = 1$, $P = 1$, and $c_4 = 1$. Obviously, the block's carry out C_{out} is one. Now consider when the same block performing its next operation with $C_{in} = 0$ and $P = 1$. Therefore, the full adders must generate $c_4 = 0$. The carry ripple through the full adders will take some time. Meanwhile, the AND-OR logic will "postpone" generating the C_{out} until c_4 is resolved. The Fredkin carry skip logic in Fig. 10 passes C_{in} to C_{out} whenever $P = 1$, regardless of c_4. The time savings can be significant as block sizes increase.

The full adder in Fig. 5 requires five FGs, but does not generate the required propagate signal p_i. Both the full adder's sum bit s_i and the carry bit c_{i+1} are generated by passing through three FGs. This full adder cannot generate the propagate signal without at least one additional FG. The full adder in Fig. 8 requires four FGs, and generates the propagate signal p_i required by carry skip adder circuits. Both the full adder's sum bit s_i and the carry bit c_{i+1} are generated by passing through three FGs. To avoid fanout, the full adder in Fig. 8 is modified to become the circuit in Fig. 9. An additional FG is required to replicate the propagate signal. The delay to generate s_i is three FG delays (from x_i to s_i), and the delay to generate p_i is four FG delays (from x_i to p_i). The delay for c_{i+1} would appear to be five, but the full adder carry in signal is not a controlled input, instead it is the fourth FG's control signal. Therefore, the maximum datapath length is four FG delays from x_i to c_{i+1}.

A B bit full adder requires $4B$ and $5B$ FGs using the circuits in Fig. 8 and Fig. 9, respectively. The B input AND gate requires $B-1$ FGs. Therefore, a B bit carry skip adder block requires $5B$ and $6B$ FGs by using the full adder circuits in Fig. 8 and Fig. 9, respectively.

Consider the B carry skip adder block in Fig. 10 generating a block carry out C_{out} generate via carry ripple through the full adders. (The full adder circuit in Fig. 9 is used throughout the remainder of this paper.) The least significant full adder requires a path delay of four FGs to generate c_1 from the addends. Then, the carry "ripples" through the subsequent full adders with a path delay of two FGs per bit. Finally, C_{out} is generated by the FG in the left of Fig. 10. Therefore, the delay to generate block carry out C_{out} (via ripple) with a B bit carry skip adder block is

$$d_{C_{ripple}}(B) = 2B + 3. \tag{4}$$

Consider the B carry skip adder block in Fig. 10 generating a block carry out C_{out} generate via a carry skip. Each full adder requires a path delay of four FGs to generate p_i from the addends. (All full adders generate p_i simultaneously.) Then, all propagate signals for the carry skip adder block are combined with a B bit AND gate with delay given by (1). Finally, C_{out} is generated by the FG in the left of Fig. 10.

$$d_{C_{skip}}(B) = \lceil \log_2 B \rceil + 5. \tag{5}$$

The total (worst case) delay T_{fixed} of an N bit carry skip adder with fixed block size B is the sum of the ripple carry delay through the first carry skip adder block, the skip delays through the intermediate blocks, and the ripple carry delay through the last block, or

$$T_{fixed} = (2B+3) + \left(\frac{N}{B} - 2\right)(\lceil \log_2 B \rceil + 5) + (2B+3). \tag{6}$$

However, the assumption

$$\lceil \log_2 B \rceil \approx B/2 \tag{7}$$

is valid for the small block sizes B applicable to carry skip adder designs. Thus, (6) can be written

$$T_{fixed} = 3B + \frac{5N}{B} + \frac{N}{2} - 4. \tag{8}$$

Minimizing T_{fixed} with respect to block size B gives

$$3 - \frac{5N}{B^2} = 0 \text{ or } B_{opt} = \sqrt{\frac{5N}{3}}. \tag{9}$$

Substituting (9) into (6) gives the shortest delay for a fixed block size carry skip adder

$$T_{fixed} = \frac{N}{2} + 7.75\sqrt{N} - 4. \tag{10}$$

3.4. Variable block carry skip adder

Varying the size of the carry skip blocks can reduce the total worst case delay, since carries generated or absorbed in the adder center have shorter data paths [7]. Without loss of generality, the first and last carry skip blocks are b bits wide, and the carry skip adder is divided into t blocks, where t is even. Assuming that carry skip block sizes are

$$b \;\; b+1 \;\; \ldots \;\; b+\frac{t}{2}-1 \;\; b+\frac{t}{2}-1 \;\; \ldots b+1 \;\; b. \tag{11}$$

Summing the number of bits in the blocks, equating to N, and rearranging gives

$$b = \frac{N}{t} - \frac{t}{4} + \frac{1}{2}. \tag{12}$$

The total (worst case) delay $T_{variable}$ of an N bit carry skip adder with the variable block sizes is the sum of the ripple carry delay through the first carry skip adder block, the skip delays through the intermediate blocks, and the ripple carry delay through the last block. Assuming the variable block sizes in (11), the total delay is

$$T_{variable} = 2(2b+3) + 2\sum_{k=b+1}^{b+\frac{t}{2}-1} (\lceil \log_2 k \rceil + 5). \quad (13)$$

Inserting (7) into (13), and rearranging gives

$$T_{variable} = 4b - 4 + 5t + \sum_{k=b+1}^{b+\frac{t}{2}-1} k. \quad (14)$$

Using the identity

$$\sum_{k=X}^{Y} k = \frac{Y(Y+1) - X(X-1)}{2} \quad (15)$$

to simplify (14) yields

$$T_{variable} = 4b - 4 + 5t + \frac{\left(b+\frac{t}{2}-1\right)\left(b+\frac{t}{2}\right) - (b)(b+1)}{2} = \frac{t^2}{8} + \frac{19t}{4} + \frac{bt}{2} + 3b - 4 \quad (16)$$

Inserting (12) into (16), and collecting terms gives

$$T_{variable} = \frac{17}{4}t - \frac{5}{2} + \frac{3N}{t} + \frac{N}{2}. \quad (17)$$

The optimal number of blocks is found with

$$\frac{\partial T_{variable}}{\partial t} = \frac{17}{4} - \frac{3N}{t^2} = 0 \text{ or } t_{opt} = \frac{2\sqrt{51}}{17}\sqrt{N}. \quad (18)$$

Therefore, the optimal variable block size carry skip adder has delay

$$T_{variable} = \frac{N}{2} + \sqrt{51}\sqrt{N} - \frac{5}{2}. \quad (19)$$

4. Conclusions

The focus of the work described here centered on finding good architectures for adder circuits based on the FG. The new full-adder cells have the desirable properties of minimizing critical path (in terms of unit FG delays) and of avoiding fanout. A CSA structure is proposed and analyzed in terms of technolgy independent implementations. It is necessary to perform technolgy independent analyses because quantum or optical logic implementations are not available.

To verify the functionality of our circuits, they were implemented in standard CMOS using the circuit in Fig. 11 for a FG. We heavily emphasize the fact that a FG based implementation in CMOS is not an optimal or even desirable way to implement addition circuits, rather we used this as a means to validate our architectures. FG full adders were simulated using Fig. 11. The timing numbers measure the elapsed time between application of a test vector and the rising edge of the outputs. The left and right numbers in Table I are the timing information for proposed FG full adders in Fig. 8 and Fig. 9, respectively.

TABLE I
TIMING OF FULL ADDER FIG. 8 AND FIG. 9
(TIMES ARE IN UNIT FREDKIN GATE DELAYS)

	x_i	y_i	c_i
s_i	1.72/1.83	1.56/2.06	1.00/1.00
p_i	1.72/1.89	1.28/1.44	-
c_{i+1}	1.94/1.94	1.89/1.56	1.00/1.39

5. References

[1] C. H. Bennett and D. P. DiVincenzo, "Quantum Information and Computation", *Nature*, vol. 404, pp. 247-255, March 2000.

[2] C. H. Bennett, "Logical Reversibility of Computation", *IBM J. Research and Development*, pp. 525-532, November 1973.

[3] E. Fredkin and T. Toffoli, "Conservative Logic", *Intl. J. Theoretical Physics*, vol. 21, nos. 3-4, pp. 219-253, 1982.

[4] R. Landauer, "Irreversibility and Heat Generation in the Computing Process", *IBM J. Research and Development*, vol. 3, pp. 183-191, July 1961.

[5] R.C. Minnick, P.T. Bailey, R.M. Sandfort, and W.L. Semon, "Cascade realizations of magnet bubble logic using a small set of primitives", *IEEE Trans. Computers*, pp. 215-217, 1975.

[6] G. J. Milburn, "Quantum Optical Fredkin Gate", *Physical Review Letters*, vol. 62, no. 18, pp. 2124-2127, May 1989.

[7] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, 2000.

[8] M Perkowski and P Kerntopf, "Reversible Logic", invited tutorial, Proc, EURO-MICRO,Warsaw, Poland Sept 2001.

[9] M. Perkowski, et al, "Regularity and Symmetry as a Base for Efficient Realization of Reversible Logic Circuits", *Proc.Intl. Workshop on Logic Synthesis*, pp. 90-95, June 2001.

[10] P.D. Picton, "Fredkin gates as the basis for comparison of different logic designs", *Synthesis and Optimisation of Logic Systems*, IEE, London, 1994.

[11] T. Sasao and K. Kinoshita, "Conservative logic elements and their universality", *IEEE Trans. Computers*, vol. 28, no. 9, pp. 682-685, 1979.

[12] M. A. Thornton, R. Drechsler and D. M. Miller, *Spectral Techniques in VLSI CAD*, Kluwer Academic Publishers, Boston, 2001.

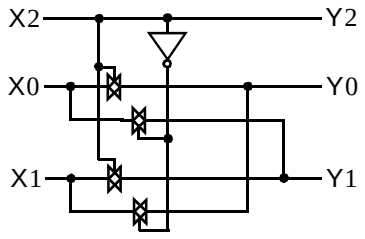

Fig. 11. Fredkin gate CMOS implementation