INVITED PAPER *Special Section on Frontiers of Superconductive Electronics*

Comparisons of Synchronous-Clocking SFQ Adders

Naofumi TAKAGI[†a)] ***and*** **Masamitsu TANAKA**[†]**,** ***Members***

SUMMARY Recent advances of superconducting single-flux-quantum (SFQ) circuit technology make it attractive to investigate computing systems using SFQ circuits, where arithmetic circuits play important roles. In order to develop excellent SFQ arithmetic circuits, we have to design or select their underlying algorithms, called hardware algorithms, from different point of view than CMOS circuits, because SFQ circuits work by pulse logic while CMOS circuits work by level logic. In this paper, we compare implementations of hardware algorithms for addition by synchronous-clocking SFQ circuits. We show that a set of individual bit-serial adders and Kogge-Stone adder are superior to others.

key words: *single-flux-quantum (SFQ) circuit, adder, hardware algorithm*

1. Introduction

Superconducting single-flux-quantum (SFQ) circuit technology is expected to be a next-generation circuit technology which enables ultra high-speed computation with ultra low-power consumption [1]. Several simple SFQ microprocessor LSIs with more than ten thousand Josephson junctions have been fabricated using 2 μm process technology [2]–[4]. 1 μm process technology with five interconnection layers is now available [5]. The increase of the wiring layers combined with the passive transmission line (PTL) technology, which provides ballistic transmission of an SFQ pulse on superconducting wiring [6], [7], further increases the circuit integration density. (For the recent progress of SFQ circuit technology, see e.g., [4], [8].) The progress of SFQ circuit technology makes it attractive to investigate computing systems using SFQ circuits. In such computing systems, arithmetic circuits play important roles, and hence, excellent SFQ arithmetic circuits are desired. In the development of excellent arithmetic circuits, design or selection of their underlying algorithms, called hardware algorithms, is crucial.

In SFQ digital circuits, an SFQ pulse is used for representing the logic value 0 or 1. There are two ways for representing the logic value, 'synchronous-clocking' method and 'dual-rail' method, and the former is widely used today. In the synchronous-clocking method, synchronizing clocks are introduced and the two logic values are represented by the absence and the presence of a pulse on a data line in a clock period, respectively. In synchronous-clocking SFQ circuits, each logic gate is a clocked gate and has a function of latch. Therefore, additional non-computing elements, i.e., D flip-flops (DFFs), have to be introduced to make the length, i.e., the number of elements, of all paths equal and thus to provide for efficient pipelining. This feature is completely different from semiconductor circuits, such as CMOS circuits, which work by level logic. Therefore, we have to design or select hardware algorithms for arithmetic circuits suitable for synchronous-clocking SFQ circuit implementation from a different point of view than semiconductor circuits.

In this paper, we consider synchronous-clocking SFQ adders. Adders are the most fundamental arithmetic circuits. There are several hardware algorithms for addition. We consider their implementations by synchronous-clocking SFQ circuits, and compare the adders with each other with respect to the number of required elements, the latency and the throughput. Note that the throughput is an important criterion because of the pipelined processing. We show that a set of individual bit-serial adders and a Kogge-Stone adder are superior to others.

In Sects. 2 and 3, we will review synchronous-clocking SFQ circuits and hardware algorithms for addition, respectively. In Sect. 4, we will consider implementations of the hardware algorithms by synchronous-clocking SFQ circuits, and will make comparison. In Sect. 5, we will conclude this paper.

2. Synchronous-Clocking SFQ Circuits

The basic component of SFQ digital circuits is a superconducting loop with Josephson junctions. A single flux quantum, which appears as a voltage pulse (SFQ pulse), is used as the carrier of information. The width of an SFQ pulse is several pico-seconds and the height is about 1 mV. Therefore, the energy consumed for switching is much smaller and the speed for switching is higher than those of CMOS circuits.

There are two ways for representing the logic value, 0 or 1. One is 'synchronous-clocking' method and the other is 'dual-rail' method where two signal lines are used for representing the two logic values for each. Since it is more widely used today, we consider synchronous-clocking method in this paper. In the synchronous-clocking method, synchronizing clocks are introduced and the two logic values are represented by the absence and the presence of a pulse on a data line in a clock period, respectively, as shown in Fig. 1(a). Each logic gate is a clocked gate and has a clock input as well as data inputs as shown in Fig. 1(b). Each logic gate has a function of latch.

Manuscript received August 1, 2009.
Manuscript revised December 7, 2009.

[†]The authors are with the Department of Information Engineering, Nagoya University, Nagoya-shi, 464-8603 Japan.
a) E-mail: ntakagi@is.nagoya-u.ac.jp

DOI: 10.1587/transele.E93.C.429

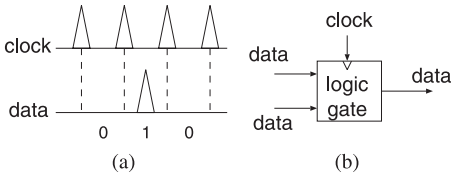


Fig. 1 Synchronous-clocking method. (a) Representation of logic values. (b) Logic gate.

In synchronous-clocking SFQ circuits, since each logic gate is a clocked gate, additional non-computing elements, i.e., D flip-flops (DFFs), have to be introduced to make the length, i.e., the number of elements, of all paths from the inputs to each logic gate equal. Then, efficient pipelining is achieved. This feature is completely different from semiconductor circuits, such as CMOS circuits, which work by level logic.

We have to mention that in SFQ circuits, a large fan-out causes relatively large delay than in CMOS circuits, because we have to split an SFQ pulse. For a fan-out of k, we need a tree of $k-1$ splitters with height $\lceil \log_2 k \rceil$.

3. Hardware Algorithms for Addition

In this section, we review hardware algorithms for addition designed to be implemented by semiconductor circuits. Let us consider addition of two unsigned binary integers, $X = [x_{n-1}x_{n-2}\cdots x_0]$ and $Y = [y_{n-1}y_{n-2}\cdots y_0]$. Let the sum be $S = [s_n s_{n-1} s_{n-2}\cdots s_0]$.

3.1 Bit-Serial Adder

A bit-serial adder serially adds X and Y by taking operand bit-pairs, (x_i, y_i)'s, at a rate of one per cycle and producing sum bits, s_i's, at the same rate, from the least significant bit to the most significant one, i.e., from $i = 0$ to $n-1$. In each cycle, s_i and the carry to the next cycle, c_{i+1}, are calculated from x_i, y_i, and the carry from the previous cycle, c_i, as $s_i = x_i \oplus y_i \oplus c_i$, and $c_{i+1} = x_i y_i + c_i(x_i + y_i)$ (or $= x_i y_i + c_i(x_i \oplus y_i)$). A circuit implementing this one-bit addition is called a full adder (FA). The essential part of a serial adder consists of a full adder and a delay element (DFF) which stores the carry for a cycle as shown in Fig. 2.

3.2 Ripple Carry Adder

The simplest parallel adder is a ripple carry adder. An n-bit ripple carry adder is constructed by cascading n full adders in series as shown in Fig. 3. The carry output of each FA is connected with the carry input of the FA of the next upper position. (At the least significant position, a half adder (HA) is used instead of a full adder, because there is not a carry input.) The number of logic gates is proportional to n. The delay is proportional to n.

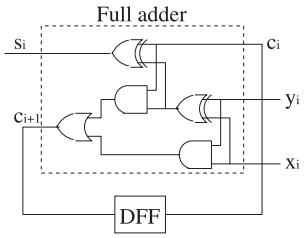


Fig. 2 A bit-serial adder.

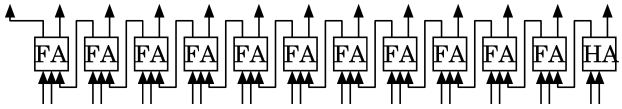


Fig. 3 A ripple carry adder.

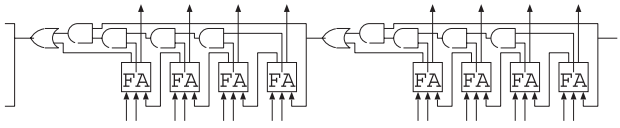


Fig. 4 A part of a carry skip adder.

3.3 Carry Skip Adder

In a ripple carry adder, a carry propagates through the i-th FA when $x_i \oplus y_i = 1$. Hereafter, p_i denotes $x_i \oplus y_i$. A carry propagates through a block of consecutive FAs, when all p_i's in the block are 1. This condition is called the carry propagation condition of the block. A carry skip adder is obtained by partitioning a ripple carry adder into several blocks of FAs and attaching a carry skip circuit to each block (except the most and the least significant ones) as shown in Fig. 4. A carry skip circuit detects the carry propagation condition of the block and lets the carry from the next lower block bypass the block when the condition holds.

We can reduce the worst delay to being proportional to $\sqrt{n}$, by letting the block size be proportional to $\sqrt{n}$. Note that the length of the longest path is still proportional to n, and is a bit longer than that of a ripple carry adder. The number of logic gates is proportional to n.

3.4 Carry Select Adder

A carry select adder is obtained by partitioning a ripple carry adder into several blocks, calculating the two sub-sums in each block (except the least significant one), i.e., one with assuming a carry input of 0 and the other with assuming a carry input of 1, in parallel, and selecting the correct one using multiplexers (MUXs) controlled by the actual carry output from the next lower block. Figure 5 shows a part of a carry select adder.

We can reduce the worst delay to being proportional to $\sqrt{n}$, by letting the block size be proportional to $\sqrt{n}$. The number of logic gates is proportional to n. The maximum fan-out of logic gates is proportional to $\sqrt{n}$.

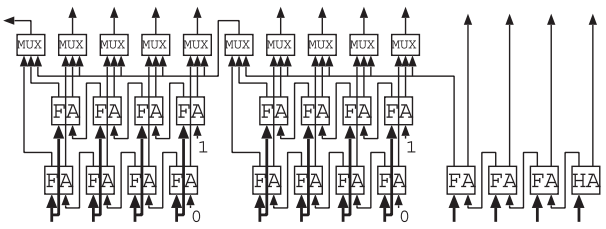


Fig. 5 A carry select adder.

3.5 Carry Look-Ahead Adder and Parallel Prefix Adders

The carry, c_{i+1}, produced at the i-th position is calculated as $c_{i+1} = x_i y_i + c_i(x_i \oplus y_i)$. This states that a carry is generated if both x_i and y_i are 1, and an incoming carry is propagated if p_i is 1. Therefore, letting $x_i y_i$ be g_i, we have $c_{i+1} = g_i + p_i c_i$. $g_i = 1$ is the carry generation condition at the i-th position. Substituting $g_{i-1} + p_{i-1}c_{i-1}$ for c_i, we get $c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$. Recursive substitutions yield $c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots + p_i p_{i-1} \cdots p_1 g_0$. A carry look-ahead adder calculates c_{i+1}'s at all bit positions in parallel according to this formula.

The idea of carry look-ahead can be generalized to parallel prefix addition. When we know g and p signals of two consecutive groups of bit positions $g_{j,i}$, $p_{j,i}$ for $i, \cdots, j$ and $g_{k,j+1}$, $p_{k,j+1}$ for $j+1, \cdots, k$, we can easily compute g and p for the larger group $i, \cdots, k$; $g_{k,i} = g_{k,j+1} + p_{k,j+1} g_{j,i}$, $p_{k,i} = p_{k,j+1} p_{j,i}$. Different ways of subdividing the operands into bit groups lead to a family of parallel prefix adders. Figures 6(a), (b), and (c) show structures of Kogge-Stone (K-S) [9], Ladner-Fisher (L-F) [10], and Brent-Kung (B-K) [11] adder, respectively. Note that $c_{i+1} = g_{i,0}$ and $s_i = p_i \oplus c_i = p_i \oplus g_{i-1,0}$.

The delay of these three adders is proportional to $\log n$. That of L-F adder is the smallest. The number of logic gates of B-K adder is proportional to n, while that of K-S and L-F adder is proportional to $n \log n$. Although the number of logic gates of K-S adder is larger than the other two, fan-outs of the logic gates in K-S adder are fewer than the others.

4. Synchronous-Clocking SFQ Adders

We consider implementations of the adders shown in the previous section by synchronous-clocking SFQ circuits. We compare them with respect to the number of required elements, the latency and the throughput. The number of required elements is the total of the number of logic gates and that of the inserted DFFs. Recall that we have to insert DFFs in order to make the length, i.e., the number of elements, of all paths from the inputs to each logic gate equal. The latency is the number of clock cycles required for performing an n-bit addition. The throughput is the rate of number of the performed n-bit additions per clock cycle.

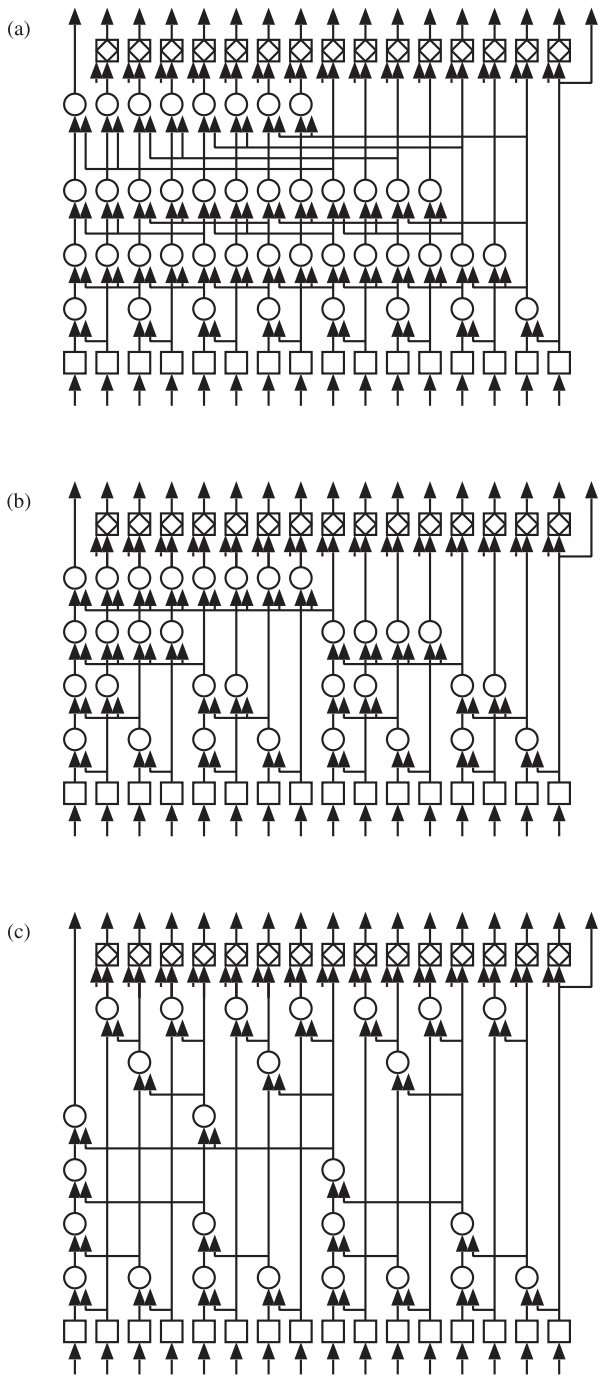


Fig. 6 Parallel prefix adders. (a) Kogge-Stone adder. (b) Ladner-Fisher adder. (c) Brent-Kung adder. The bottom row of each adder calculates p_i and g_i from operand bits x_i and y_i. Computations of $p_{k,i}$ and $g_{k,i}$, denoted by white circles, are carried out upward, and the sum bits s_i are produced by the top row. Lines for bringing p_i's from the bottom row to the top row are not shown, in order to prevent each figure from being complicated.

4.1 An SFQ Bit-Serial Adder

We can construct an SFQ bit-serial adder using an SFQ full adder as shown in Fig. 7. The OR for producing the carry output c_{i+1} (See Fig. 2) is realized by just merging data lines

using a confluence buffer (CB), because its two inputs cannot be 1 simultaneously. A DFF is inserted for making the length of all paths from x_i and y_i to c_{i+1} equal. The carry output is connected to the carry input. We do not need a DFF on this feed-back loop because each logic gate has a function of latch.

The latency for n-bit addition is $n + 1$. The throughput is $\frac{1}{n+1}$. The number of required elements is $O(1)$.

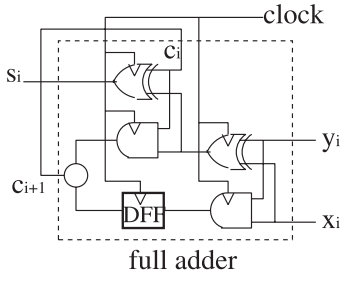


Fig. 7 An SFQ bit-serial adder.

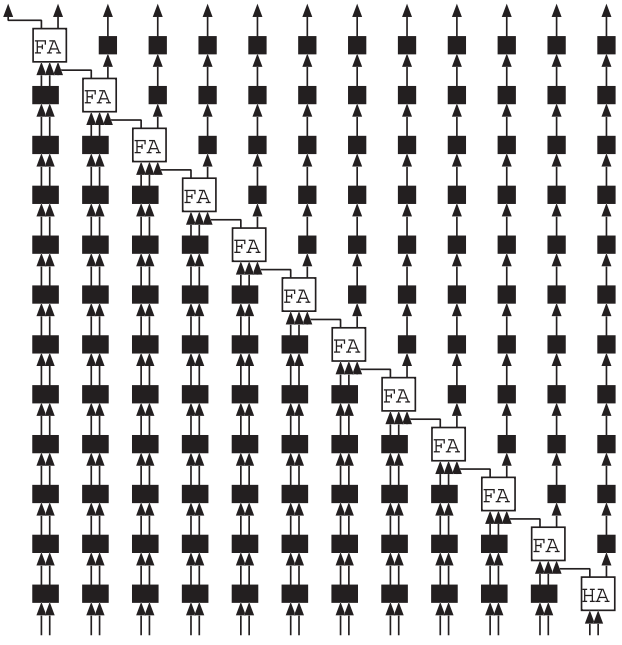


Fig. 8 An SFQ ripple carry adder.

4.2 An SFQ Ripple Carry Adder

For implementing a ripple carry adder by synchronous-clocking SFQ circuits, we need to insert $O(n^2)$ DFFs to make the length of all paths equal as shown in Fig. 8. The black boxes denote the inserted DFFs.

The latency is $n + 1$. The throughput is 1. The number of required elements is $O(n^2)$.

4.3 An SFQ Carry Skip Adder

An SFQ carry skip adder requires more logic gates and DFFs than an SFQ ripple carry adder, because the length of its longest path is larger. Its latency is also longer. Note that we have to insert DFFs on the bypass lines in order to make the length of all paths from the inputs to each logic gate equal, and therefore, a carry signal cannot be forwarded faster through a bypass.

4.4 An SFQ Carry Select Adder

For implementing a carry select adder consisting of $O(\sqrt{n})$ blocks by synchronous-clocking SFQ circuits, we need to insert $O(n\sqrt{n})$ DFFs as shown in Fig. 9. The latency is $O(\sqrt{n})$, because the length of the longest path is proportional to $\sqrt{n}$. Since there are $O(\sqrt{n})$ logic gates with fan-out of $O(\sqrt{n})$, a lot of splitters are required for forming fan-out trees, and they increase the clock period. The number of required elements is $O(n\sqrt{n})$

4.5 SFQ Parallel Prefix Adders

An SFQ K-S, L-F and B-K adder, all require $O(n \log n)$ additional DFFs as shown in Fig. 10, because the length of the each longest path is proportional to $\log n$. Their latency is $O(\log n)$. A K-S and an L-F adder are superior to a B-K adder in SFQ implementation, though they require more

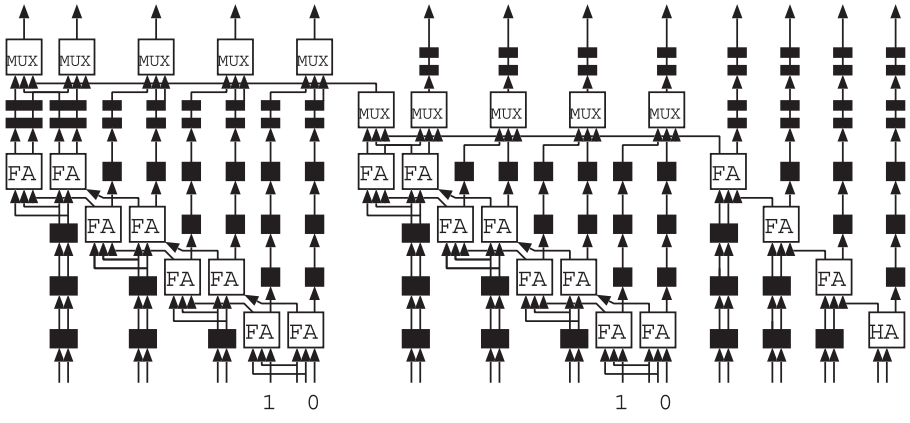


Fig. 9 An SFQ carry select adder.

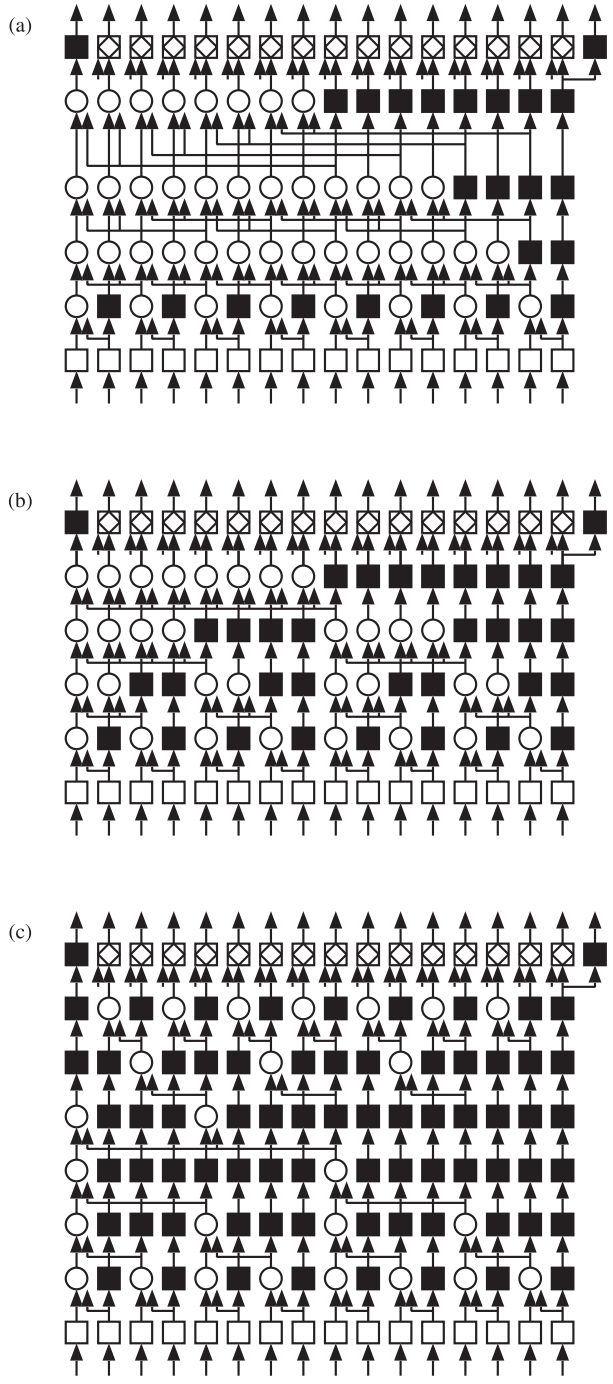


Fig. 10 SFQ parallel prefix adders. (a) Kogge-Stone adder. (b) Ladner-Fisher adder. (c) Brent-Kung adder.

logic gates (computing elements) than the latter, because the length of their longest path is shorter than that of the latter. A K-S adder is superior to an L-F adder, because fan-outs of logic gates in it are fewer than those in the latter.

4.6 Comparisons of Synchronous-Clocking SFQ Adders

Table 1 shows comparisons of synchronous-clocking SFQ adders. In summary, a K-S adder seems to be the best as an SFQ parallel adder [12]. A bit-serial adder is also attractive for SFQ implementation. Using n individual bit-serial adders in parallel, we can obtain the same throughput as parallel adders, with much fewer hardware.

Table 1 Comparisons of SFQ adders.

adder	latency	through-put	# elements	Max. fan-out
bit-serial	$n+1$	$\frac{1}{n+1}$	$O(1)$	$O(1)$
ripple carry	$n+1$	1	$O(n^2)$	$O(1)$
carry skip	$O(n)$	1	$O(n^2)$	$O(n)$
carry select	$O(\sqrt{n})$	1	$O(n\sqrt{n})$	$O(\sqrt{n})$
Kogge-Stone	$O(\log n)$	1	$O(n \log n)$	$O(1)$
Ladner-Fisher	$O(\log n)$	1	$O(n \log n)$	$O(n)$
Brent-Kung	$O(\log n)$	1	$O(n \log n)$	$O(1)$

5. Conclusion

We have considered implementations of adders by synchronous-clocking SFQ circuits and have compared them. In CMOS implementation of parallel adders, there exists a trade-off between computation speed and amount of hardware in general. Namely, faster is larger. On the other hand, in synchronous-clocking SFQ implementation of parallel adders, adders with small latency consist of fewer elements. Among various parallel adders, Kogge-Stone adder seems to be most suitable for SFQ implementation.

Utilization of several individual bit-serial adders in parallel is also attractive. This solution provides the same throughput as parallel adders with much less hardware, although its latency is a bit large.

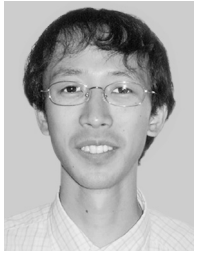
For multiplication, division and square-rooting, bit-serial circuits with systolic array structure seem to be suited for SFQ implementation [13], [14].

References

[1] K.K. Likharev and V.K. Semenov, "RSFQ logic/memory family: A new Josephson-Junction technology for sub-terahertz-clock-frequency digital systems," IEEE Trans. Appl. Supercond., vol.1, no.1, pp.3–28, March 1991.
[2] M. Tanaka, T. Kondo, N. Nakajima, T. Kawamoto, Y. Yamanashi, Y. Kamiya, A. Akimoto, A. Fujimaki, H. Hayakawa, N. Yoshikawa, H. Terai, Y. Hashimoto, and S. Yorozu, "Demonstration of a single-flux-quantum microprocessor using passive transmission lines," IEEE Trans. Appl. Supercond., vol.15, no.2, pp.400–404, June 2005.
[3] Y. Yamanashi, M. Tanaka, A. Akimoto, H. Park, Y. Kamiya, N. Irie, N. Yoshikawa, A. Fujimaki, H. Terai, and Y. Hashimoto, "Design and implementation of a pipelined bit-serial SFQ microprocessor, CORE1β," IEEE Trans. Appl. Supercond., vol.17, no.2, pp.474–477, June 2007.
[4] A. Fujimaki, M. Tanaka, T. Yamada, Y. Yamanashi, H. Park, and N. Yoshikawa, "Bit-serial single flux quantum microprocessor CORE," IEICE Trans. Electron., vol.E91-C, no.3, pp.342–349, March 2008.
[5] T. Satoh, K. Hinode, H. Akaike, S. Nagasawa, Y. Kitagawa, and M. Hidaka, "Characteristics of Nb/AlOx/Nb junctions fabricated in planarized multi-layer Nb SFQ circuits," Physica C, vol.445-448, pp.937–940, 2006.

[6] S.V. Polonsky, V.K. Semenov, and D.F. Schneider, "Transmission of single-flux-quantum pulses along superconducting microstrip lines," IEEE Trans. Appl. Supercond., vol.3, no.1, pp.2598–2600, 1993.
[7] T. Yamada, H. Ryoki, A. Fujimaki, and S. Yorozu, "Flexible superconducting passive interconnects with 50-Gb/s signal transmissions in single-flux-quantum circuits," Jpn. J. Appl. Phys. Pt. 1, vol.45, no.2A, pp.752–757, 2006.
[8] N. Yoshikawa, "Recent development and perspective of ultra-high-speed microprocessors using single-flux-quantum circuits," IEICE Trans. Electron. (Japanese Edition), vol.J91-C, no.3, pp.183–193, March 2008.
[9] P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Trans. Comput., vol.C-22, no.8, pp.786–793, Aug. 1973.
[10] R.E. Ladner and M.J. Fischer, "Parallel prefix computation," J. ACM, vol.27, no.4, pp.831–838, Oct. 1980.
[11] R.P. Brent and H.T. Kung, "A regular layout for parallel adders," IEEE Trans. Comput., vol.C-31, no.3, pp.260–264, March 1982.
[12] P. Bunyk and P. Litskevitch, "Case study in RSFQ design: Fast pipelined parallel adder," IEEE Trans. Appl. Supercond., vol.9, no.2, pp.3714–3720, June 1999.
[13] K. Obata, M. Tanaka, Y. Tashiro, Y. Kamiya, N. Irie, K. Takagi, N. Takagi, A. Fujimaki, N. Yoshikawa, H. Terai, and S. Yorozu, "Single-flux-quantum integer multiplier with systolic array structure," Physica C, vol.445-448, pp.1014–1019, 2006.
[14] M. Tanaka, K. Obata, K. Takagi, N. Takagi, A. Fujimaki, and N. Yoshikawa, "A high-throughput single-flux-quantum floating-point serial divider using the signed-digit representation," IEEE Trans. Appl. Supercond., vol.19, pp.653–656, June 2009.

Naofumi Takagi received the B.E., M.E., and Ph.D. degrees in information science from Kyoto University, Kyoto, Japan, in 1981, 1983, and 1988, respectively. He joined Department of Information Science, Kyoto University, as an instructor in 1984 and was promoted to an associate professor in 1991. He moved to Department of Information Engineering, Nagoya University, Nagoya, Japan, in 1994, where he has been a professor since 1998. His current interests include computer arithmetic, hardware algorithms, and logic design. He received Japan IBM Science Award and Sakai Memorial Award of the Information Processing Society of Japan in 1995.



Masamitsu Tanaka received the M.E. and Ph.D. degrees in electronics, and information electronics from Nagoya University, Nagoya, Japan, in 2003 and 2006, respectively. He was a JSPS Research Fellow from 2005 to 2007 at Department of Quantum Engineering, Nagoya University. He joind Department of Information Engineering as a research fellow in 2007, where he has been a designated assistant professor since 2009. His research interests include the ultra-fast computing using single-flux-quantum circuits and logic design methodologies. He is a member of the IEEE and the JSAP.