Logic Functions Realized Using Clockless Gates for Rapid Single-Flux-Quantum Circuits

Takahiro Kawaguchi
Graduate School of Informatics
Kyoto University
Kyoto, 606-8501, Japan
kawaguti@lab3.kuis.kyoto-u.ac.jp

Kazuyoshi Takagi
Graduate School of Engineering
Mie University
Mie, 514-8507, Japan
ktakagi@arch.info.mie-u.ac.jp

Naofumi Takagi
Graduate School of Informatics
Kyoto University
Kyoto, 606-8501, Japan
takagi@i.kyoto-u.ac.jp

Abstract— Superconducting rapid single-flux-quantum (RSFQ) circuit is a promising candidate for circuit technology in the post-Moore era. RSFQ digital circuits operate with pulse logic and are usually composed of clocked gates. We have proposed clockless gates, which can reduce the hardware amount of a circuit drastically. In this paper, we first discuss logic functions realizable using clockless gates. We show that all the 0-preserving functions can be realized using the existing clockless gates, i.e., AND, OR, and NIMPLY (not-imply) gates. We then propose a circuit architecture composed of both clocked gates and clockless gates.

I. Introduction

Superconducting rapid single-flux-quantum (RSFQ) circuit technology is a promising candidate for circuit technology in the post-Moore era [1]. It uses quantized voltage pulses and magnetic flux quanta for digital data generation and memorization. A Josephson junction, an active device in RSFQ circuit technology, switches quickly in a few picoseconds and dissipates a little switching energy less than 10^{-19} J [2]. In RSFQ digital circuits, logic values of 1 and 0 are usually represented by the presence and the absence of a data pulse, respectively, between two consecutive clock pulses. Therefore, ordinary logic gates have a clock input and are called *clocked gates*. Most RSFQ circuits consist of clocked gates [3, 4, 5, 6, 7].

A clocked gate has a latching function, which stores the data until a clock pulse arrives. The latching function of clocked gates leads to a deep-pipelined circuit, called a gate-level-pipelined circuit [8]. The number of pipeline stages, i.e., the pipeline depth, of a gate-level-pipelined circuit is the same as its logic level. A deep pipeline achieves a very high clock frequency. However, it requires a large circuit area because of a large number of delay flip-flops (DFFs) for path-balancing [9].

We have proposed clockless gates in [10]. A clockless gate is a logic gate without a clock input and does not have a latching function. When clockless gates are used instead of some of the clocked gates in a circuit, the pipeline depth and the area occupied by a clock distribution network are reduced. The reduction of the pipeline depth leads to a fewer DFFs for path-balancing and a smaller circuit area. A clockless AND gate and a clockless NIMPLY gate was proposed in [10]. A NIMPLY (not-imply) gate is an AND gate with one inverted input, which implements $x \wedge \neg y$. A confluence buffer [1], which merges a pulse from its inputs to its output, can be used as a clockless OR gate.

A clockless gate realizes only a 0-preserving logic function, which outputs 0 from the input value of all 0s, because it cannot produce a pulse from no pulse arrival. It is known that any logic function generated from a set of 0-preserving logic functions is 0-preserving [11, 12]. Thus, a circuit composed of clockless gates realizes 0-preserving logic functions.

In this paper, we first discuss two-input 0-preserving functions. Next, we show that all the 0-preserving functions can be generated from {AND, OR, NIMPLY}. We then propose a circuit architecture composed of both clocked gates and clockless gates. It implements a circuit with a register and a combinational circuit. The function of the register is realized by the latching function of the clocked gates. The combinational circuit is realized by the logic function of the clocked gates and that of the clockless gates.

The rest of this paper is organized as follows. Clockless gates are explained in Section II. Logic functions realized using clockless gates are discussed in Section III. A circuit architecture composed of both clocked gates and clockless gates is proposed in Section IV. The paper is concluded in Section V.

II. Clockless gates

In RSFQ digital circuits, logic values of 1 and 0 are usually represented by the presence and the absence of a data pulse, respectively, between two consecutive clock pulses. Ordinary logic gates for RSFQ circuits are clocked gates. A clocked gate produces an output pulse, if any, from a pulse arrival at its clock input as shown in Fig. 1(a). A clocked gate has a latching function, which stores the data until a clock pulse arrives. In the CONNECT cell library

for RSFQ circuits [13], there exist a clocked buffer, i.e., a delay flip-flop (DFF), a clocked inverter and clocked AND, OR, XOR, NOR and NAND gates.

A clockless gate is a logic gate without a clock input. It produces an output pulse, if any, from a pulse arrival at its data input as shown in Fig. 1(b). A clockless gate has timing constraints that require close pulse arrivals at its inputs. We proposed a clockless AND gate and a clockless NIMPLY gate in [10]. A confluence buffer [1], which merges a pulse from its inputs to its output, can be used as a clockless OR gate. Clockless AND and OR gates with higher input skew tolerance were proposed in [14].

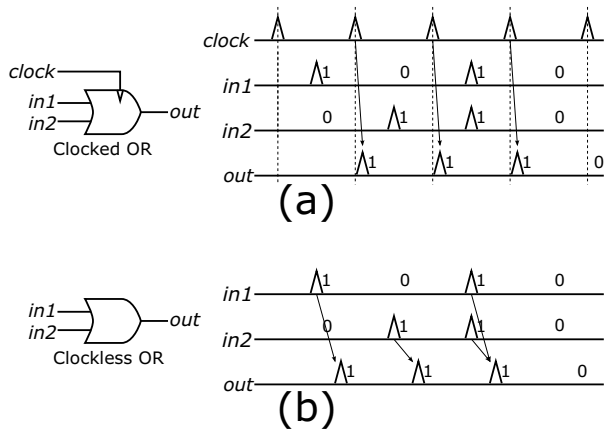

Fig. 1. (a) Clocked OR gate and (b) clockless OR gate

A circuit composed of only clocked gates has a gate-level-pipeline whose pipeline depth is the same as its logic level. It requires a large number of DFFs for path-balancing [9]. For example, the 4-bit carry-lookahead adder that we designed includes 20 clocked logic gates and 25 DFFs for path-balancing, and its logic level was 6 as shown in Fig. 2(a). Hence, the clock distribution network has 45 sinks. Its pipeline depth is 6.

When we designed a 4-bit carry-lookahead adder with two pipeline stages using clocked gates and clockless gates, it includes 12 clocked logic gates, 10 clockless gates and 4 DFFs as shown in Fig. 2(b). Its clock distribution network has 16 sinks. A few DFFs for path-balancing and a small clock distribution network leads to a small area.

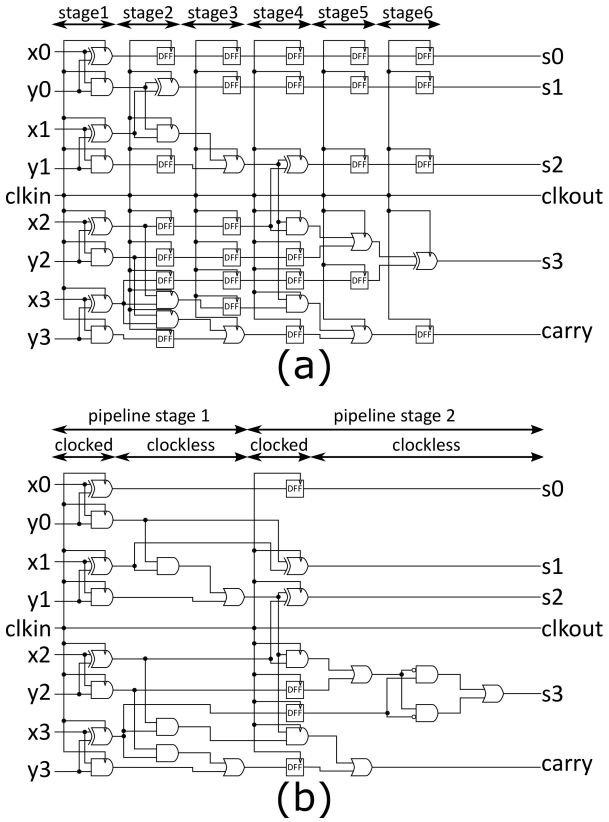

Fig. 2. (a) 4-bit carry-lookahead adder using clocked gates with 6 pipeline stages and (b) 4-bit carry-lookahead adder using both clocked gates and clockless gates with 2 pipeline stages

III. Logic functions realized using clockless gates

A clockless gate produces an output pulse, if any, from a pulse arrival at its data input. Therefore, a clockless gate realizes a 0-preserving function, which outputs 0 from the input value of all 0s. It is known that a logic function generated from a set of 0-preserving logic functions is 0-preserving [11, 12].

We consider clockless gates with two-inputs. The two-input 0-preserving functions are AND ($x \wedge y$), OR ($x \vee y$), NIMPLY ($x \nrightarrow y = x \wedge \neg y$ and $x \nleftarrow y = \neg x \wedge y$) and XOR ($x \oplus y$).

In this section, we first discuss two-input 0-preserving functions. Then, we show that all the 0-preserving functions can be generated from {AND, OR, NIMPLY}.

A. *Two-input 0-preserving functions*

From {AND}, OR, NIMPLY and XOR cannot be generated. From {OR}, AND, NIMPLY and XOR cannot be generated. From {XOR}, AND, OR and NIMPLY cannot be generated.

From {NIMPLY}, AND can be generated as follows:

$$x \wedge y = x \nrightarrow (x \nrightarrow y). \tag{1}$$

Tables I and II show all the two-input logic functions that can be generated from {NIMPLY}. Note that $0 = x \nrightarrow x$. In Table I, the entry of row x and column $x \nrightarrow y$ is $x \nrightarrow (x \nrightarrow y) = x \wedge y$. OR and XOR cannot be generated from {NIMPLY}.

Table III lists two-input 0-preserving functions generated from {AND}, {OR}, {NIMPLY} and {XOR}.

TABLE I
TWO-INPUT LOGIC FUNCTIONS GENERATED FROM {NIMPLY} (ROW $\nrightarrow$ COLUMN)

$\nrightarrow$	0	x	y	$x \wedge y$	$x \nrightarrow y$	$x \nleftarrow y$
0	0	0	0	0	0	0
x	x	0	$x \nrightarrow y$	$x \nrightarrow y$	$x \wedge y$	x
y	y	$x \nleftarrow y$	0	$x \nleftarrow y$	y	$x \wedge y$
$x \wedge y$	$x \wedge y$	0	0	0	$x \wedge y$	$x \wedge y$
$x \nrightarrow y$	$x \nrightarrow y$	0	$x \nrightarrow y$	$x \nrightarrow y$	0	$x \nrightarrow y$
$x \nleftarrow y$	$x \nleftarrow y$	$x \nleftarrow y$	0	$x \nleftarrow y$	$x \nleftarrow y$	0

TABLE II
TWO-INPUT LOGIC FUNCTIONS GENERATED FROM {NIMPLY} (ROW $\wedge$ COLUMN)

$\wedge$	0	x	y	$x \wedge y$	$x \nrightarrow y$	$x \nleftarrow y$
$x \nrightarrow y$	0	$x \nrightarrow y$	0	0	$x \nrightarrow y$	0
$x \nleftarrow y$	0	0	$x \nleftarrow y$	0	0	$x \nleftarrow y$

Next, we consider two-input logic functions generated from the subsets of {AND, OR, NIMPLY, XOR} including two functions, i.e., {AND, OR}, {AND, NIMPLY}, {AND, XOR}, {OR, NIMPLY}, {OR, XOR} and {NIMPLY, XOR}.

Since AND and OR are monotonically increasing functions, the logic functions generated from {AND, OR} are monotonically increasing [11, 12]. Therefore, from {AND, OR}, NIMPLY and XOR cannot be generated.

TABLE III
TWO-INPUT LOGIC FUNCTIONS GENERATED FROM THE SUBSETS OF {AND, OR, NIMPLY, XOR} INCLUDING ONE FUNCTION

set	AND	OR	NIMPLY	XOR
{AND}	Yes	No	No	No
{OR}	No	Yes	No	No
{NIMPLY}	Yes	No	Yes	No
{XOR}	No	No	No	Yes

The two-input logic functions generated from {AND, NIMPLY} are the same as those generated from {NIMPLY}.

From {AND, XOR}, OR and NIMPLY can be generated as follows:

$$x \vee y = (x \wedge y) \oplus (x \oplus y), \tag{2}$$

$$x \nrightarrow y = x \oplus (x \wedge y). \tag{3}$$

From {OR, NIMPLY}, XOR can be generated as follows:

$$x \oplus y = (x \nrightarrow y) \vee (x \nleftarrow y). \tag{4}$$

AND can be generated by Eq.(1).

From {OR, XOR}, NIMPLY and AND can be generated as follows:

$$x \nrightarrow y = (x \vee y) \oplus y, \tag{5}$$

and AND,

$$x \wedge y = (x \vee y) \oplus (x \oplus y). \tag{6}$$

From {NIMPLY, XOR}, OR can be generated as follows:

$$x \vee y = x \oplus (x \nleftarrow y). \tag{7}$$

AND can be generated by Eq.(1)

Table IV lists two-input 0-preserving functions generated from the subsets of {AND, OR, NIMPLY, XOR} including two functions.

From any subset of {AND, OR, NIMPLY, XOR} including three functions, e.g., {AND, OR, NIMPLY}, all the two-input 0-preserving functions can be generated. Recall that AND, OR and NIMPLY exist as clockless gates for RSFQ digital circuits.

B. *Logic functions generated from {AND, OR, NIMPLY}*

As previously mentioned, from {AND, OR, NIMPLY}, only 0-preserving logic functions can be generated. Here, we prove that all the 0-preserving functions can be generated from {AND, NIMPLY, OR}.

TABLE IV
TWO-INPUT LOGIC FUNCTIONS GENERATED FROM THE SUBSETS OF {AND, OR, NIMPLY, XOR} INCLUDING TWO FUNCTIONS

set	AND	OR	NIMPLY	XOR
{AND, OR}	Yes	Yes	No	No
{AND, NIMPLY}	Yes	No	Yes	No
{AND, XOR}	Yes	Yes	Yes	Yes
{OR, NIMPLY}	Yes	Yes	Yes	Yes
{OR, XOR}	Yes	Yes	Yes	Yes
{NIMPLY, XOR}	Yes	Yes	Yes	Yes

A value of the input variables X of a logic function is expressed by

$$i = \{\forall x_l \in X_1, \forall x_m \in X_0, x_l = 1, x_m = 0\} \tag{8}$$

where X_1 is the set of the input variables that are 1 and X_0 is the set of the input variables that are 0. We assume i is not all 0s, in other words, X_1 is not an empty set.

A 0-preserving function that outputs 1 from an input value i is expressed by

$$g_i = \begin{cases} 1 & (\text{if } X = i) \\ 0 & (\text{the other cases}) \end{cases} \tag{9}$$

g_i can be generated by

$$g_i = (\bigwedge_{\forall x_l \in X_1} x_l) \nrightarrow (\bigvee_{\forall x_m \in X_0} x_m). \tag{10}$$

An arbitrary 0-preserving function is expressed by

$$f_I = \begin{cases} 1 & (\text{if } X = i \in I) \\ 0 & (\text{the other cases}) \end{cases} \tag{11}$$

where I is a set of arbitrary input values except all 0s.

f_I can be generated by

$$f_I = \bigvee_{\forall i \in I} g_i. \tag{12}$$

□

Note that {AND, XOR}, {OR, NIMPLY}, {OR, XOR} and {NIMPLY, XOR} are the minimal sets of the two-input 0-preserving functions from which all the 0-preserving functions can be generated.

IV. A CIRCUIT ARCHITECTURE COMPOSED OF CLOCKED AND CLOCKLESS GATES

When a synchronous system such as a microprocessor is realized by a CMOS circuit, the calculation (processing) in one cycle is performed by a combinational circuit sandwiched between registers. Here, we propose a circuit architecture using both clocked gates and clockless gates which implements a circuit that integrates a register on the input side and a combinational circuit. As shown in Fig.3, it is composed of a column of clocked gates and a succeeding block of clockless gates. The function of the register is realized by the latching function of the clocked gates in the column and the function of the combinational circuit is realized by the logic function of these clocked gates and the logic function of the clockless gates in the block. In the architecture, the first level consists of clocked gates and the second and subsequent levels consist of clockless gates.

The 4-bit carry-lookahead adder in Fig. 2 were designed in the architecture. Each of the pipeline stages is composed of a column of clocked gates and a succeeding block of clockless gates.

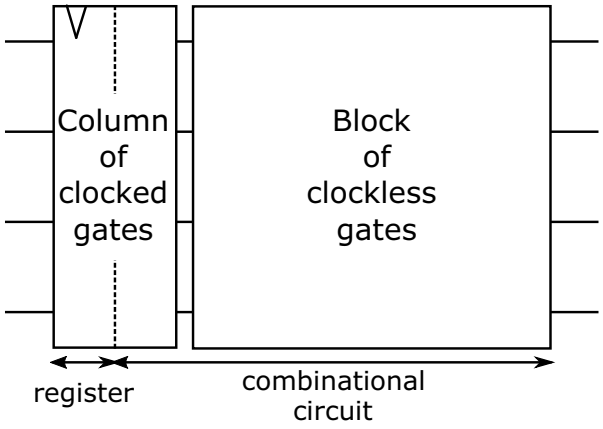

Fig. 3. Proposed circuit architecture composed of a column of clocked gates and a succeeding block of clockless gates

Since only a 0-preserving function can be realized using clockless gates, it is necessary to develop a new logic design method for designing a circuit in this architecture. A circuit in this architecture can be designed by the following four steps. (1) For each non-0-preserving logic function in a given combinational logic, we find an input value for which the logic function outputs 0 and replace the logic function into a 0-preserving logic function by negating some of its input variables based on the found input value. (2) We construct an and-inverter graph (AIG) from the combinational logic composed of the 0-preserving logic functions. (3) We map the primary-input nodes into DFFs and clocked inverters, and map the AIG nodes into AND, NIMPLY and OR gates. (4) We merge logic gates with DFFs and/or clocked inverters and replace them with clocked gates, if possible.

In step (1), given a combinational logic, we replace each non-0-preserving logic function into a 0-preserving logic function by nagating some of its input variables. The given combinational logic is expressed by $CL = \{F, X\}$, where F is a set of logic functions and X is a set of input variables. For each $f \in F$, if f is non-0-preserving, we

find an input value for which f outputs 0. The found input value is expressed by

$$i_f = \{\forall x_l \in X_1, \forall x_m \in X_0, x_l = 1, x_m = 0\} \qquad (13)$$

where $X_1 \cup X_0$ is X. Then, we replace f into a 0-preserving logic function f' by negating every variables in X_1. The set of input variables of f' is expressed by $Y_{f'} = X_0 \cup X_1'$ where X_1' is a set of the negated input variables, $\{y = \neg x_l | x_l \in X_1\}$. As a result, the given combinational logic is replaced by the combinational logic $CL' = \{F', Y\}$, where F' is a set of 0-preserving logic functions and $Y = \bigcup Y_{f' \in F'}$ is a set of the input variables of CL'. $|Y|$ is greater than or equal $|X|$ and less than or equal $2|X|$. The large size of Y can result in the large circuit area.

In step (2), we construct an AIG representing CL'. An AIG is a directed acyclic graph whose nodes have either 0 or 2 incoming edges [15, 16, 17]. A node with no incoming edges is a primary input (PI). A node with 2 incoming edges represents a two-input AND. An edge is either complemented or not. A complemented edge indicates an inverter. An edge with no destination node represents a primary output (PO). The combinational logic of an arbitrary Boolean network can be represented by an AIG [18].

In step (3), we first map every PI node in the constructed AIG into a DFF or a clocked inverter. A PI node representing an input variable of CL is mapped into a DFF. A PI node representing a negated input variable in step (1) is mapped into a clocked inverter. Next, we map every AIG node into an AND, a NIMPLY or an OR gate from the PIs to the POs. An unmapped node whose inputs are connected to a PI or a mapped node is mapped into a logic gate by adopting the following rules as shown in Fig. 4. (a) A node with two non-complemented incoming edges is mapped into an AND gate. (b) A node with one non-complemented incoming edge and one complemented incoming edge is mapped into a NIMPLY gate and the complemented incoming edge is replaced by a non-complemented edge. (c) A node with two complemented incoming edges is mapped into an OR gate and the complemented incoming edges are replaced by non-complemented edges. Its outgoing edges are inverted. In other words, a non-complemented outgoing edge is replaced by a complemented edge, and vice versa. This mapping procedure is iterated until all the nodes are mapped. When the mapping is done, all the PO edges in the mapping result are non-complemented because the PO edges in the AIG represent 0-preserving logic functions.

In step (4), we merge AND, NIMPLY and OR gates with DFFs and/or clocked inverters and replace them with clocked gates, if possible, as shown in Fig. 5. By using clocked gates, e.g., clocked AND, OR and EXOR gates, a circuit is optimized in terms of the numbers of clocked gates and clockless gates and the maximum level

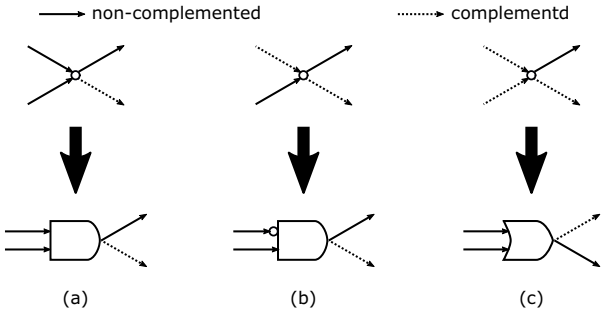

Fig. 4. Mapping an AIG node into a logic gate: (a) mapping a node with two non-complemented incoming edges into an AND, (b) mapping a node with one non-complemented incoming edge and one complemented incoming edge into a NIMPLY and (c) mapping a node with two complemented incoming edges into an OR with inverted outgoing edges

of clockless gates. A logic gate whose inputs are connected to DFFs and/or clocked inverters is a candidate to be merged.

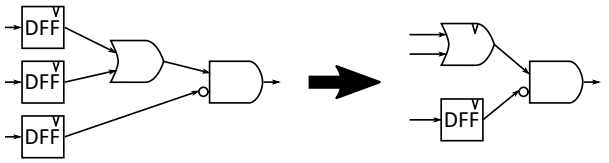

Fig. 5. Merging an OR gate with DFFs and replacing them with a clocked OR gate

V. Conclusion

A clockless gate realizes only a 0-preserving function because it cannot produce a pulse from no pulse arrival. Therefore, the logic functions realized using clockless gates are 0-preserving. We have discussed two-input 0-preserving functions. Then, we have shown all the 0-preserving functions can be generated from a set of the existing clockless gates, AND, OR and NIMPLY.

We proposed a circuit architecture composed of a column of clocked gates and a block of clockless gates. In the proposed architecture, a clocked inverter is important because only a 0-preserving function can be realized using clockless gates. As future work, we will develop a logic design method and a logic synthesis tool for designing a circuit in the proposed architecture.

References

[1] Konstantin K Likharev and Vasilii K Semenov. RSFQ logic/memory family: A new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *Applied Superconductivity, IEEE Transactions on*, 1(1):3–28, 1991.

[2] D Scott Holmes, Andrew L Ripple, and Marc A Manheimer. Energy-efficient superconducting computing-power budgets and requirements. *IEEE Transactions on Applied Superconductivity*, 23(3):1701610–1701610, 2013.

[3] Soheil Nazar Shahsavani, Ting-Ru Lin, Alireza Shafaei, Coenrad J Fourie, and Massoud Pedram. An integrated row-based cell placement and interconnect synthesis tool for large SFQ logic circuits. *IEEE Transactions on Applied Superconductivity*, 27(4):1–8, 2017.

[4] Soheil Nazar Shahsavani and Massoud Pedram. A minimum-skew clock tree synthesis algorithm for single flux quantum logic circuits. *IEEE Transactions on Applied Superconductivity*, 29(8):1–13, 2019.

[5] Naveen Kumar Katam and Massoud Pedram. Logic optimization, complex cell design, and retiming of single flux quantum circuits. *IEEE Transactions on Applied Superconductivity*, 28(7):1–9, 2018.

[6] Masamitsu Tanaka, Hiroyuki Akaike, Akira Fujimaki, Yuki Yamanashi, Nobuyuki Yoshikawa, Shuichi Nagasawa, Kazuyoshi Takagi, and Naofumi Takagi. 100-GHz single-flux-quantum bit-serial adder based on 10-niobium process. *IEEE Transactions on Applied Superconductivity*, 21(3):792–796, 2011.

[7] Koki Ishida, Masamitsu Tanaka, Takatsugu Ono, and Koji Inoue. Towards ultra-high-speed cryogenic single-flux-quantum computing. *IEICE Transactions on Electronics*, 101(5):359–369, 2018.

[8] Yoshio Kameda, Shinichi Yorozu, and Yoshihito Hashimoto. Automatic single-flux-quantum (SFQ) logic synthesis method for top-down circuit design. In *Journal of Physics: Conference Series*, volume 43, page 1179. IOP Publishing, 2006.

[9] Ghasem Pasandi and Massoud Pedram. PBMap: A path balancing technology mapping algorithm for single flux quantum logic circuits. *IEEE Transactions on Applied Superconductivity*, 29(4):1–14, 2018.

[10] Takahiro Kawaguchi, Masamitsu Tanaka, Kazuyoshi Takagi, and Naofumi Takagi. Demonstration of an 8-bit SFQ carry look-ahead adder using clockless logic cells. In *2015 15th International Superconductive Electronics Conference (ISEC)*, pages 1–3. IEEE, 2015.

[11] William Wernick. Kimio ibuki, kenzi naemura, and akihiro nozaki. general theory of complete sets of logical functions. electronics and communications in japan, vol. 46 no. 7 (for 1963, pub. 1964), pp. 5565. *Journal of Symbolic Logic*, 37(2):416417, 1972.

[12] Emil L Post. *The Two-Valued Iterative Systems of Mathematical Logic.(AM-5), Volume 5*, volume 5. Princeton University Press, 2016.

[13] Yuki Yamanashi, Toshiki Kainuma, Nobuyuki Yoshikawa, Irina Kataeva, Hiroyuki Akaike, Akira Fujimaki, Masamitsu Tanaka, Naofumi Takagi, Shuichi Nagasawa, and Mutsuo Hidaka. 100 GHz demonstrations based on the single-flux-quantum cell library for the 10 ka/cm^2 Nb multi-layer process. *IEICE transactions on Electronics*, 93(4):440–444, 2010.

[14] Sergey V Rylov. Clockless dynamic SFQ and gate with high input skew tolerance. *IEEE Transactions on Applied Superconductivity*, 29(5):1–5, 2019.

[15] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 532–535. IEEE, 2006.

[16] Ian Shrier and Robert W Platt. Reducing bias through directed acyclic graphs. *BMC medical research methodology*, 8(1):70, 2008.

[17] Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification*, pages 24–40. Springer, 2010.

[18] Robert K Brayton. The decomposition and factorization of boolean expressions. In *Proc. Int. Symp. Circ. Sys.(ISCAS 82) Rome*, 1982.