A Timing Fault Model and an Efficient Timing Fault Simulation Method for Rapid Single-Flux-Quantum Logic Circuits

Shogo Nakamura\textsuperscript{1}, Kazuyoshi Takagi\textsuperscript{1}, Nobutaka Kito\textsuperscript{2}, Naofumi Takagi\textsuperscript{3}
\textsuperscript{1}Graduate School of Engineering, Mie University
\textsuperscript{2}School of Engineering, Chukyo University
\textsuperscript{3}Graduate School of Informatics, Kyoto University

Abstract. We examine digital behavior of faults caused by a change of the order of the pulse arrivals in Rapid Single-Flux-Quantum (RSFQ) logic circuits. Based on the timing fault model, we present a flow of automatic test pattern generation for testing digital RSFQ chips. As the test pattern generation repeatedly executes fault simulation, the scalability depends on computational cost of the simulation. We propose an efficient fault simulation method based on two ideas. First, we share computation for the fault-free circuit and faulty circuits as much as possible. Secondly, we exploit the pipelined behavior of RSFQ logic circuits to suppress computation for unnecessary signals. We have implemented our simulation method and obtained evaluation results to show that our method is effective for large circuits.

1. Introduction

Rapid Single-Flux-Quantum (RSFQ) logic circuits have been researched actively in recent years. An RSFQ logic circuit can operate at high frequency over several tens of GHz with low-power consumption. Due to diminishing growth in performance of a CMOS circuit, RSFQ circuits have been attracted attention as alternatives to CMOS circuits. Unlike common CMOS circuits, operation of RSFQ circuits is based on pulse logic. Each logic gate of RSFQ circuits is clocked. An input logic value to an RSFQ gate is represented by pulse arrival between two adjacent clock pulses. That is, if a pulse arrives within a period, it represents logic-1. If a pulse does not arrive, it represents logic-0. A gate includes several superconducting loops and holds a logical state. If a loop traps a single flux quantum, it represents state-1. Otherwise, it represents state-0. Because of this logic representation as well as high speed operation, small disturbance of pulse timing by physical factors can change logical behavior of circuits.

A fabricated chip is tested to check the functionality. The test is done by feeding prepared patterns to the circuit and observing the responses. The prepared patterns are called test patterns. Expected outputs for the input test patterns are also prepared. If actual outputs differ from the expected outputs, we can find out that a fault exists in the circuit under test.

Though EDA tool suites for CMOS circuits include test pattern generators, they are not directly applicable to RSFQ circuits. In testing RSFQ circuits, it has often been the case that ad hoc test patterns are fed to the chip and the outputs are observed. Using an ad hoc method, we cannot guarantee to detect faults. This problem becomes more serious as circuits become...
larger. Length of a test pattern sequence is also an important factor because it is directly linked to the cost for testing.

To introduce new testing flow for RSFQ circuits, we need a systematic test pattern generation method that generates short test patterns which can detect as many faults as possible. In this paper, we deal with digital behavior of RSFQ gates. We focus on the changes of pulse arrival order caused by fabrication variations or operation environments. In RSFQ circuits, the behavior of a gate depends on the order of the pulse arrival. Our viewpoint is common with a recent research proposing the circuit description method that allows designers to specify the order of the pulse arrival to each input port of gates[1]. The change of pulse arrival caused by timing disturbance can modify the temporal behavior of the gate. The modification can cause error in the logic function of the circuit.

In our study, to identify such faults, we define a fault model using a set of inequalities in timing. Then, based on the model, we develop a test pattern generation method using fault simulation. In our test pattern generation method, we need iterative execution of fault simulation which is computationally expensive. Therefore, we suggest optimization methods of fault simulation to reduce computation time of the test pattern generation.

2. Preliminaries and related works
A test of fabricated digital circuit is done by feeding prepared test patterns to the circuit, observing the responses, and comparing the actual outputs and the expected outputs for the test patterns. Ad hoc test generations are time-consuming and do not guarantee to detect faults. Therefore, we need automatic test pattern generation methods.

By automatic test pattern generation, the circuit under test and the faults to be assumed are given, and, a test pattern sequence with the expected output sequence is obtained. In general, test pattern generation flow consists of a simulation-based part and an algorithm-based part. For testing large circuits with large numbers of assumed faults, the simulation-based part generates test patterns that detect most of the faults. Here, test patterns are generated randomly and the detection ability is evaluated using fault simulation. The most effective pattern is added to the current test pattern sequence. This process is repeated until the ratio of detected fault becomes sufficient.

In fault simulation, the number of faults detected by the input pattern is calculated by simulating a miter circuit shown in Figure 1. A miter circuit consists of a fault-free circuit and faulty circuits. A faulty circuit reproduces behavior of the circuit with an assumed fault. An input pattern detects a fault if outputs of the fault-free circuit and the faulty circuits differ for the pattern. A larger circuit requires more assumed faults to be introduced and hence the miter circuit becomes large. Accordingly, the computational cost of fault simulation becomes very large.

Concurrent fault simulation method[2] has been proposed to improve the efficiency of fault simulation, where calculation results of the fault-free circuit are reused in calculation for faulty circuits. In our study, we also focus on the common structures of the circuits. For testing RSFQ circuits, we have to deal with the operation principle of RSFQ circuits which differs from that of CMOS circuits. We incorporate ideas of logic simulation algorithms for RSFQ circuits[1].

In the algorithm-based part, test patterns for the faults that are not detected by simulation are generated. A basic method for this part is D-algorithm[3], which iterates value assignments to wires and propagates their effects to outputs of the circuit under test. Improved methods such as FAN[4] and PODEM[5] algorithms are also well-known. Another approach is Boolean SAT-based test pattern generation, which has become effective with the progress of SAT solvers.

Related works on verification and testing of RSFQ circuits can be found recently[6, 7]. In [6], methods to insert test points and scan chains for RSFQ circuits are discussed. In [7], path delay faults are focused and a test pattern generation method for them is proposed. Our motivation
in this paper is to give a new approach in modelling delay faults to cover a wide range of digital errors. In the measurement of fabricated chips, incomplete functionality is often explained, for example, as that arrival of a certain signal at a certain gate is too early or too late relative to the clock signal. Timing of data pulses as well as clock pulses at each gate can be disturbed by various causes. To explain those situations, we focus on the order of pulse arrival at each gate. Based on the notion of the pulse arrival order, we build a timing fault model, design a test pattern generation method, and propose an efficient fault simulation method.

3. Timing fault model for RSFQ digital circuits

We suppose the situation that the circuit under test responds to logical stimulus and the outputs can be captured, and the logical functionality of the circuit has to be confirmed with respect to the design. The circuit under test consists of clocked logic gates, but does not contain feedback loops. The aim is to test circuit modules on fabricated chips in laboratories.

In this paper, we propose a test pattern generation method to detect timing faults of RSFQ circuits. Because a test pattern generation method needs a fault model (e.g. stuck-at fault model), we need to define a fault model related to pulse timing. We use fault simulation in our test pattern generation method. To reproduce fault behavior in logic simulation, an unambiguous fault model that can identify a fault caused by changes of pulse timing is needed.

We focus on timing faults of RSFQ circuits. Signal delay in RSFQ circuits depends on various factors such as the fabrication process and the operating environment. Because pulse timing at all clocked gates can affect the logical behavior, and because the clock frequency is high, timing faults in RSFQ circuits are very common.

Considering the circuit behavior composed of clocked gates as described in the previous section, timing faults in RSFQ circuits can be broken down to discrepancies of pulse arrival timing between the design and the fabricated chip. Our basis is the property that if the arrival order of pulses at any clocked gate has changed for some reason, logical behavior of the circuit can change and that becomes a logical error. What matters is the order, and the amount of delay does not matter with the logical behavior as far as the order is preserved. From this observation, we define a timing fault of an RSFQ circuit as the state that “the arrival order of pulses at a clocked gate is discrepant between the design and the circuit under test.”

Let $i_0^{G3}(t)$ represent time that a pulse arrives at input pin $i_0$ of gate $G3$ at the $t$-th clock cycle. We omit the gate specification, e.g. $i_0(t)$, when appropriate.

A fault corresponds to an arrival order of pulses, which is different to the design, at a clocked gate. The set of all faults to be assumed is defined from this property. Note that generating all assumed faults is easy because this property is local to each gate. For example, let us consider timing faults at a clocked gate in a faulty circuit as shown in Figure 1. Let us assume that

Figure 1. A miter circuit
this gate is a clocked AND gate designed to work with concurrent-flow clocking in the circuit under test. Here, we describe the pulse arrival order that clock comes earlier than \( i_0 \) and \( i_1 \), and the clock of the next cycle follows, as “\( \text{clock}(t) < i_0(t), i_1(t) < \text{clock}(t + 1) \).” Note that the relationship with the clock pulse of the next cycle depends on the clock period, so the clock period should be given. Timing faults with respect to this gate corresponds to the following “wrong” pulse arrival orders:

\[
\begin{align*}
& i_0(t) < \text{clock}(t) < i_1(t) < \text{clock}(t + 1), \text{ or,} \\
& i_1(t) < \text{clock}(t) < i_0(t) < \text{clock}(t + 1), \text{ or,} \\
& i_0(t), i_1(t) < \text{clock}(t) < \text{clock}(t + 1), \text{ or,} \\
& \text{clock}(t) < i_0(t) < \text{clock}(t + 1) < i_1(t), \text{ or,} \\
& \text{clock}(t) < i_1(t) < \text{clock}(t + 1) < i_0(t), \text{ or,} \\
& \text{clock}(t) < \text{clock}(t + 1) < i_0(t), i_1(t), \text{ or,} \\
& i_0(t) < \text{clock}(t) < \text{clock}(t + 1) < i_1(t), \text{ or,} \\
& i_1(t) < \text{clock}(t) < \text{clock}(t + 1) < i_0(t).
\end{align*}
\]

(Each of the third and sixth case can further be divided into the cases that \( i_0(t) < i_1(t) \) and \( i_1(t) < i_0(t) \). However, their equivalence is obvious from the behavior of an AND gate and we treat them unified.) Timing faults for gates with other functions or with clock-follow-data clocking are defined similarly. The assumed fault set for the circuit under test is the union of fault sets for all gates. Hence, the assumed fault set can be instantly generated once the circuit netlist accompanied with the order of pulse arrival at each gate is given.

4. Automatic test pattern generation for RSFQ digital circuits

In this section, we describe a test pattern generation method for timing fault detection. For a given circuit under test, we first obtain the set of assumed faults as described in the previous section. (Alternatively, the set of assumed faults can be given by the designer.) Our method generates a test pattern sequence, i.e., a sequence of primary input patterns to detect the faults. We assume that at most one fault can occur in the circuit. We also assume that the order of pulse arrival changes by at most one clock cycle. For example, for the order \( a(t) < b(t) \) in the design, we assume the case of \( b(t) < a(t) \) but do not further assume the case of \( b(t + 1) < a(t) \). We aim at detecting all faults that are not redundant. (A redundant fault is a fault with which the circuit output is the same as that of the fault-free circuit for any input pattern.) Quality of the test pattern sequence is measured by the length, i.e. the number of patterns.

Let us consider a sample circuit \( C_1 \) in Figure 2 and a timing fault at gate \( G_3 \) for example. \( C_1 \) is designed with the concurrent-flow clocking scheme and works as a 4-stage pipelined circuit.

Pulses at \( i_{0G_3}^G \) and \( i_{1G_3}^G \) are designed to arrive between the corresponding clock pulse and the next clock pulse at \( \text{clock}^{G_3} \), i.e., \( \text{clock}^{G_3}(t) < i_{0G_3}^G(t), i_{1G_3}^G(t) < \text{clock}^{G_3}(t + 1) \) is the right order.

Correct operation at gate \( G_3 \) is depicted in Figure 3. The input signal \( i_{0G_3}^G \) and \( i_{1G_3}^G \) at clock cycle \( n \) is correctly processed and a pulse at the output \( o_{0G_3}^G \) is produced following the clock pulse in the next cycle at \( \text{clock}^{G_3} \). Figure 4 is a timing chart when the pulse arrival at \( i_{1G_3}^G \) is delayed, so that pulse arrival of the succeeding clock pulse at \( \text{clock}^{G_3} \) is earlier, i.e., the case \( \text{clock}^{G_3}(t) < i_{0G_3}^G(t) < \text{clock}^{G_3}(t + 1) < i_{1G_3}^G(t) \). The reason could be due to the fabrication process, or due to the operating environment such as the amount of bias current. Here the logical behavior of the circuit has changed and the function of the whole circuit can be different to the design.

To detect this kind of timing faults, a test composed of two consecutive input patterns is effective. With two consecutive input patterns, we observe the outputs corresponding to the
two clock cycles and compare them with the correct operation. Some of the outputs can depend on input patterns of the preceding or the succeeding clock cycles, because clock cycles of the responses can be shifted. Those outputs are dealt as ‘don’t care’ with no discrepancy. Let us consider the case for our sample circuit C1 in Figure 2 and a fault depicted in Figure 4. If the values of the inputs of \( G_3 \) at \((i_0, i_1)\) are set to \((1, 1)\) followed by \((1, 0)\), the value at the output of \( G_3 \) changes. Therefore, \(((0, 1, 1, 0, 0), (0, 1, 0, 0, 0))\) for \((\text{in}_0, \text{in}_1, \text{in}_2, \text{in}_3, \text{in}_4)\) can be a test pattern sequence to detect this fault. The expected output at \((\text{out}_0, \text{out}_1)\) is \((1, 1)\) without the fault and \((0, 0)\) with the fault, meaning that the discrepancy is observable and hence the test pattern sequence detects the fault.

Test pattern generation is done for a specific circuit design. Therefore, the test pattern generation algorithm receives a circuit description data and assumed faults as inputs. In our proposed algorithm, because we use fault simulation based on the method supposed in [1], we need to give a circuit design data described using the circuit description method supposed in [1]. Our test generation method is as follows.

(i) Pairs of primary input patterns are generated in random. The patterns are referred to as \( p_0, p_1, \ldots \), in the following. Each pair is supposed to be input to the circuit under test successively. For each pair, fault simulation is executed and the set of detectable faults is calculated.

(ii) A pair \((p_0, p_1)\), which is not selected yet, with the largest number of detectable faults is picked up.

(iii) Primary input patterns are generated in random. A pattern \( p_2 \), where the number of newly detected faults by the transition \((p_2, p_0)\) (or \((p_1, p_2)\) resp.), is selected and \( p_2 \) is prepended (or appended resp.) to the sequence.

(iv) If all assumed faults are detected, finish. Otherwise, if most of the assumed faults are detected, go to Step (iv).
(v) We return to Step (iii) as far as there are newly detected faults. If not, we return to Step (ii) and repeat the same process as far as there are newly detected faults.

(vi) (Here, most of the assumed faults have been detected, or, no new faults have been detected with random patterns generated in Step (i) and Step (iii).) SAT-based algorithm is used to detect the remaining faults and finish.

It is a common scheme for test pattern generation to use random patterns followed by patterns generated by algorithmic methods. In our case, we have to focus on sequences of patterns instead of each individual pattern. We generate pairs of patterns in Step (i) and execute fault simulation using the pairs of patterns as consecutive inputs. As the circuit under test is a sequential circuit, differences of the outputs between the fault-free circuit and the faulty circuit are observed after certain clock cycles. We select the best pair greedily in Step (ii). We extend the pair to make a longer chain by prepending or appending patterns repeatedly in Step (iii), where the selection is again greedy. Step (iv) is the point to switch to an algorithmic generation method, for which we assume a SAT-based method. At Step (v), we try to extend the chain with Step (iii) if possible, or, we start over from a new pair with Step (ii) to add another chain. If no progress is possible from Step (iii) or Step (ii), we switch to the SAT-based method. Finally, we can obtain a chain of a pair as a test pattern of circuit under test and expected outputs can be calculated using logic simulation.

The number of patterns randomly generated in Steps (i) and (iii), and the fault coverage with which we proceed to SAT-based method in Step (iv) are parameters to control the flow and could be dependent on the problem size.

The SAT-based method we suppose is as follows. First, we build a miter circuit. This step is quite conventional. For given fault to detect, a fault-free circuit and a faulty circuit are constructed with common primary inputs. Each primary output of the fault-free circuit is XORed with the corresponding output of the faulty circuit, and all outputs of the XORs are ORed to make the primary output. Next, we check satisfiability of the miter circuit. Here, we have to note that the circuit under test is a sequential circuit. As we assumed that the circuit under test is feedback-free, if we fix a input test pattern, the output of the fault-free circuit after certain clock cycles is fully determined. As for the faulty circuit, the outputs can contain unknown values because the output can be affected by the values of the preceding or the succeeding clock cycle, which are not defined, due to a timing fault. If we input a pair of test patterns successively, outputs of at least one of the two clock cycles, depending on the fault, is determined. Therefore, the output of the miter circuit corresponding to the clock cycle without unknown outputs can be identified and it is described as a logic function of two consecutive primary inputs. A SAT solution of this logic function represents a pair of test patterns to detect the given fault. If the function is not satisfiable, the given fault is redundant.

5. Fault simulation of RSFQ circuits

5.1. Reducing computational cost of fault simulation

As the circuit under test grows larger, it is necessary to generate more pairs in order to search for pairs with high detection ability. Additionally, fault simulation requires iteration of logic simulation proportional to the number of assumed faults. Therefore, it is necessary to reduce the computational cost of logic simulation.

In logic simulation, inputs are fed to a circuit with the pipeline depth \( T \), and outputs are obtained after \( T \) clock cycles. Logic simulation of a circuit with \( N \) gates for \( T \) clock cycles requires \( O(TN) \) time. In this section, we propose a method that reduces the amount of calculation from both aspects of the number of gates and the depth of the circuit pipeline.

We use a miter circuit, which consists of a circuit without errors and a circuit with an error, to reproduce the fault behavior and compare the outputs. Faulty circuits are created by introducing
fault signals to clones of the correct circuit. In test pattern generation, we simulate the miter circuit for two consecutive input patterns.

Here we note that straightforward gate level simulation of the miter circuit contains unnecessary calculation. We reduce the calculation by following two ideas.

(i) We calculate only signals that are possibly being affected by the assumed fault.
(ii) We calculate only valid signals.

We describe these ideas in the following subsections. As these two ideas are orthogonal, they can be implemented simultaneously.

5.2. Recalculating only signals affected by faults

A fault makes outputs of gates where the signal values are different from the fault-free circuit. In a combinational circuit, signals depending on the fault insertion point break down in a chain reaction. Therefore, differences of behavior between fault-free circuit and faulty circuits are in subcircuits depending on the gate where the fault is introduced. In other words, the fault-free circuit and faulty-circuits behave exactly the same except for the subcircuits. It is not necessary to recalculate the common behavior.

In our optimization method, we aim at recalculating only the signals affected by faults. The gate-level logic simulation algorithm proposed in [1] needs a topologically sorted list of gates as an input of the algorithm. The basic implementation strategy is, deriving the set of gates in the subcircuit depending on each assumed fault, and providing a topologically sorted list of the gates to the logic simulation algorithm. To refer results of the fault-free circuit in calculation of faulty circuits, we create a gate correspondence table between the fault-free circuit and faulty-circuits. Because a combinational circuit can be seen as a directed acyclic graph (DAG), fanout cone of gate $G$ can be recursively computed as the union of fanout cones of fanouts of $G$.

5.3. Calculating only valid signals

An RSFQ circuit has the property of a fine grained pipeline. Due to this property, signals propagate through gates with clock synchronization. For example, let us consider the case that pattern transition $(0, 0, 1, 1, 1) \rightarrow (0, 0, 0, 0, 1)$ is input to the sample circuit $C1$ shown in Figure 2. When $n$ signals are input to a combinational circuit, the signals are latched to a maximum of $n$ stages. After three clock cycles, signals generated by propagating the first input pattern are latched as $(0, 0, 1, 1, 1)$ at $G3$, $G4$, $G5$, $G6$, and $G7$. They are output to the next adjacent stage in the next clock cycle. At this time, the gates $G0$, $G1$, $G2$, $G8$, and $G9$ do not latch the signals related to the two consecutive patterns. The output signal behavior of these gates has nothing to do with the test pattern output in these three clock cycles. Therefore, it is not necessary to calculate these gate behaviors in both fault-free circuit and faulty circuits.

Standard logic simulation algorithm calculates all gates in each clock cycle. However, we can omit calculation of unrelated signals to reduce computational cost of fault simulation. In a combined RSFQ circuit in test pattern generation, up to two successive stages latch valid signals in each clock cycle. The correspondence between the gate and the number of stages can be derived in advance of the circuit simulation. To reduce the amount of computation, this correspondence map is used to limit the gates calculated in each clock cycle.

6. Evaluation of proposed fault simulation method

We implemented our fault simulation method and compared it with a naïve method to evaluate the efficiency. We used combinational circuit data from ISCAS’85 dataset, which is a standard benchmark dataset widely used in several areas of digital design, including test generation, timing analysis, and technology mapping. As the designs in ISCAS’85 dataset assume (non-clocked) combinational logic gates, we converted them into RSFQ circuits with clocked logic
gates. This conversion process includes inserting D-flipflop gates to synchronize data timing, as well as simple technology mapping.

Evaluation results for circuits c17, c432, c499 and c880 in ISCAS’85 are shown in Table 1. These circuits are chosen to show the results for circuits of various sizes. We picked up faults to assume among all possible timing faults, so that the number of faults is proportional to the circuit size.

We conduct evaluation of naïve method (Naïve), our method with recalculating only the affected signals (AFF), our method with calculating only valid signals (VAL), and our method using both ideas (BOTH). Computing environment is a PC with AMD Ryzen 5 3600 Processor and RAM 16GB. Each result is an average execution time of several executions in milliseconds. It can be said from Table 1 that our method is more efficient for larger circuits, while not fully effective for very small circuits.

Table 1. Average execution time of fault simulation

| circuit | # gates | # assumed faults | Naïve [ms] | VAL [ms] | AFF [ms] | BOTH [ms] |
|---------|---------|------------------|------------|----------|----------|-----------|
| c17     | 9       | 30               | 2.84       | 2.58     | 3.06     | 2.80      |
| c432    | 1287    | 2926             | 62507.68   | 10047.73 | 12146.35 | 2501.96   |
| c499    | 2456    | 5614             | 206087.22  | 38087.91 | 13432.27 | 4427.44   |
| c880    | 5874    | 13064            | 2488094.13 | 235994.07| 90386.02 | 17682.13  |

7. Conclusions and future work
In this paper, we have proposed a digital behavior model of faults caused by a change of the order of the pulse arrivals. We also proposed an automatic test pattern generation method based on the model and optimized the method for efficient execution. We have implemented our method and obtained evaluation results that our method makes improvement of execution speed by up to 140 times. Our goal is development of an automatic test pattern generation system for practical test of fabricated RSFQ circuits. Our future work includes development of a test pattern generation system and measurement of the quality of test patterns.

Acknowledgment
This work has been supported in part by JSPS KAKENHI Grant Numbers 18K11213, 18H05211 and 19K11888.

References
[1] Kito N, Takagi K and Takagi N 2012 17th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI2012) R3-5
[2] Ulrich E G and Baker T 1998 Papers on Twenty-five years of electronic design automation pp 318-323
[3] Roth J P 1966 IBM J. Research and Development. 10 4 pp 278-91
[4] Fujiwara H and Shimono T 1983 IEEE Computer Architecture Letters, 32 12 pp 1137-44
[5] Goel P 1981 IEEE Trans. Computers. 30 3 pp 215-22
[6] Krylov G and Friedman E G 2017 IEEE Trans. Appl. Supercond. 27 8 Art no. 1302307
[7] Wang F and Sandeep G 2019 Proc. 2019 IEEE Int. Supercond. Electron. Conf.
[8] Brglez F and Fujiwara H 1985 Proc. 1985 IEEE Int. Symp. Circuits and Syst.