An RSFQ flexible-precision multiplier utilizing bit-level processing

Nobutaka Kito\(^1\) and Kazuyoshi Takagi\(^2\)

\(^1\) Department of Information Engineering, Chukyo University, Toyota, Aichi, Japan
\(^2\) Graduate School of Engineering, Mie University, Tsu, Mie, Japan

E-mail: nkito@sist.chukyo-u.ac.jp

Abstract. An RSFQ flexible-precision multiplier is proposed. The circuit can perform multiplication with specified bit-width within a predefined bit range. The calculation bit-width can be changed in every operation. When the bit-width of a calculation decreases, the latency in cycles is reduced. The proposed circuit calculates the multiplication result with bit-level processing to save the circuit area. The circuit carries out multiplication by counting pulses on a signal line. An RSFQ flexible-precision matrix multiplication circuit based on the proposed multiplier is also proposed. Its internal multipliers share many component circuits and it is implemented in a compact area.

1. Introduction

RSFQ circuits [1] are expected to realize energy-efficient high-performance computing systems for the post-Moore era. RSFQ circuits use pulse logic. Namely, voltage pulses are used for realizing logic circuits. In the logic design of RSFQ circuits, consideration for implementability of designs by the pulse logic is important.

Multiplication is an important arithmetic operation. There are various applications involving many multiplications and a part of them tolerates small error. Recently, neural networks are utilized to process various intelligent tasks. In the processing in the inference phase of neural networks, many multiplications are performed. Thus, implementing many multipliers in a chip is desired. Because the processing tolerates small error, low-precision or approximate arithmetic circuits [2] and flexible-precision circuits [3, 4] that can change calculation precision online have been proposed for CMOS circuits.

In this paper, we propose a flexible-precision multiplier for RSFQ circuits. It treats operand values whose bit-width \(v\) is in a predefined range from \(n_{\text{min}}\) to \(n\). We can specify the precision for each multiplication. For a \(v\)-bit multiplication, it uses \(2^v\) clock cycles. In other words, we can achieve higher multiplication performance when we use lower precision for processes tolerating larger error.

Parallel processing arithmetic circuits can achieve high performance with large layout area and are suitable for ALUs of microprocessors. Several RSFQ parallel processing circuits [5, 6] have been proposed. On the other hand, hardware-efficient designs utilizing bit-serial or bit-level processing [7] are suitable for massively parallel applications. Hardware efficiency is also an important factor for implementability of RSFQ circuits. The multiplier utilizes the bit-level processing proposed in our previous paper [7] for hardware efficiency. The bit-level processing
converts two operands of a multiplication fed in parallel into two bit-streams on two lines. We only use the two lines for multiplication, and we can implement the multiplier in a compact area easily. Though it performs the multiplication with an AND gate like stochastic computing [8], the multiplication is not stochastic and is carried out deterministically as a truncated multiplication. In the flexible-precision multiplier, we modify the generation circuits of bit-streams from the original ones to calculate a multiplication with reduced bit-width correctly. We insert some selectors for masking bits.

We also propose a flexible-precision matrix multiplication circuit based on the multiplier. Matrix multiplication is a computational kernel operation used commonly in a wide variety of signal processing applications and neural network applications. Because internal multipliers in the circuit share many component circuits, we can realize the circuit in a compact area.

We designed a layout of the proposed multiplier and a layout of the matrix multiplication circuit for AIST ADP2 process for evaluation purposes. They can perform 3- and 4-bit multiplication. There have been designs of RSFQ bit-serial multipliers such as [9, 10]. The number of Josephson junctions (JJs) in the layout of the proposed multiplier is smaller than those bit-serial designs. The number of JJs in the proposed matrix multiplication circuit is smaller than the number of JJs in the previously proposed circuit in [11].

2. Preliminaries

2.1. Flexible-Precision Multiplication

We consider multiplication of unsigned fixed-point numbers. In the circuits proposed in this paper, we can choose the bit-width \( v \) of calculation from a predefined range from \( n_{\text{min}} \) to \( n \). We represent the range of possible bit-width, i.e., \( n - n_{\text{min}} \), with \( f \). We represent the multiplicand \( X \) and the multiplier \( Y \) as \([0.x_1 \cdots x_v]_2\) and \([0.y_1 \cdots y_v]_2\), respectively. The multiplication result \( Z \) is \( v \)-bit fixed point number \([0.z_1 \cdots z_v]_2\). The unit in the last place (ulp) of the result is \( 2^{-v} \).

The proposed multiplier performs truncated multiplication. Namely, it discards the lower part of partial product bits as shown in Fig. 1. It sums up the upper part of bits enclosed by the solid line in the figure, whose weights are larger than \( 2^{v-1} \), and uses the bits whose weight is \( 2^{v-1} \) enclosed by the dashed lines to compensate the result. In this paper, we double the bits for compensations. In other words, we treat the weight of these bits as \( 2^{-v} \). It corresponds to the rounding to the nearest value of each partial product. Thus, we can represent the calculation result as

\[
Z = \sum_{2 \leq i+j \leq v} x_i y_j 2^{-(i+j)} + \sum_{i+j=v+1} x_i y_j 2^{-v}.
\]

2.2. RSFQ Multiplier Utilizing Bit-Level Processing

We previously proposed an RSFQ multiplier in [7]. Its inputs are the \( n \)-bit fixed-point multiplicand \( X(= [0.x_1 x_2 \cdots x_n]_2) \) and the \( n \)-bit fixed-point multiplier \( Y \) and it outputs the \( n \)-bit fixed-point result \( Z \). The bit-width of the inputs and the bit-width of the output are fixed.
We propose a flexible-precision multiplier. We can choose the bit-width \( v \) for every operation. We show its structure first. Then, we explain the steps to calculate the multiplication with it and discuss why it can perform the flexible-precision multiplication. Finally, we also propose a flexible-precision matrix multiplication circuit performing multiple multiplications in parallel.

3. Flexible-Precision Multiplier Utilizing Bit-Level Processing

We show the structure of the multiplier in Fig. 2. There are two bit-generators and each of them consists of a weighted-bits generator and a selector-and-merger. They convert the operands to sequences of bits whose period is \( 2^n - 1 \) cycles. We show the design of the bit-generator used in [7] in Fig. 3. The weighted-bits generator is a leading-one detector. It receives an \( n \)-bit vector and detects the index of the first “1” in the vector. As the index is lower, it outputs more pulses during a period of \( 2^n - 1 \) cycles. The selector-and-merger filters pulses from the weighted-bits generator according to the internal state of the non-destructive readouts (NDROs). We set the operand value as the internal state of NDROs through \( q_1, \ldots, q_n \) to convert the operand value into a sequence.

The circuit calculates logic AND of the two sequences. We obtain the result by counting the number of pulses generated with the AND gate. In the sequences, the weight of each bit is the same and is \( 2^{-n} \). If the AND gate generates a pulse for each cycle of a period of \( 2^n - 1 \) cycles, the result is \([0.11 \cdots 1]_2 = (2^n - 1) \cdot 2^{-n}\). If the AND gate generates no pulses in a period, the result is \([0.00 \cdots 0]_2\).

3.1. Structure

We show the structure of the flexible-precision multiplier in Fig. 4. \( XI \) and \( YI \) are \( n \)-bit inputs \((x_1, \ldots, x_n)\) and \((y_1, \ldots, y_n)\), respectively. We connect each bit of \( XI \) and \( YI \) to the bit with the same subscript of \( Q \) input of the corresponding selector-and-merger. \( reset_{XY} \) is an input terminal for resetting the operand values held in the selector-and-mergers. \( Z = [z_n \ldots z_1]_2 \) is the output of the \( n \)-bit pulse counter. \( reset_{counter} \) is an input terminal for resetting the pulse counter. It outputs the counted value when it receives a reset pulse. We use a binary counter for the multiplier in place of a linear feedback shift register (LFSR) in the original design in Fig. 2.

We realize the flexible-precision capability by masking the most significant \( (n-v) \)-bits of an \( n \)-bit vector fed from the binary counter. The latency in cycles to perform a \( v \)-bit multiplication is \( 2^n \). When we mask the most significant \( (n-v) \)-bits, it generates all \( v \)-bit vectors in a period.
of $2^v$ cycles while a LFSR does not guarantee this property. We mask the bits by padding 0s in $XI$ and mask the bits fed for the weighted-bits generator for $Y$ by inserting NDROs between the binary counter and the weighted-bits generator. We insert $f$ NDROs at inputs $r_1, \ldots, r_f$ of the weighted-bits generator. We name the NDRO connected to $r_i$ as NDRO$_i$ and name its reset input and its set input as $tr_i$ and $ts_i$, respectively. To configure the multiplier to $v$-bit precision, we feed pulses through $ts_1$ to $ts_{n-v}$ to set the internal state of NDRO$_1, \ldots, $ NDRO$_{n-v}$.

3.2. Operation

We show the process to perform multiplication with the circuit in Algorithm 1. First, we feed a pulse for $reset_{XY}$ to reset the operand values and feed a pulse to each of $tr$ inputs to reset the precision of calculation. Then we feed a pulse for each of $ts_1, \ldots, ts_{n-v}$ inputs to set the bit-width. Note that it is not necessary to feed pulses for $tr$ and $ts$ inputs when we do not change the bit-width from the last multiplication. We set the $v$-bit operands through $XI$ and $YI$ inputs. We feed the bits of $X$ to bit inputs with lower subscripts of $Q$ of the selector-and-merger through $XI$ and the bits of $Y$ to bit inputs with higher subscripts of $Q$ through $YI$. We pad 0s for unused bits of $XI$ and $YI$. We finally feed a pulse for the reset input of the pulse counter and obtain the result.

We discuss the operation of the multiplier carefully and explain why the circuit performs the multiplication. We can write the outputs of the weighted-bits generator, which is the same function as a leading-one detector, as $w_h = r_h \land r_{n-1} \land \cdots \land r_1 = r_h \land ( \bigwedge_{k<h} r_k )$. Thus, we can represent the output values of the selector-and-mergers as follows:

$$b_X = \bigvee_{1 \leq t \leq v} \left( x_t \land r_t \land \left( \bigwedge_{1 \leq k < t} r_k \right) \right) = \bigvee_{1 \leq t \leq v} \left( x_t \land s_t \land \left( \bigwedge_{1 \leq k < t} s_k \right) \right)$$

$$b_Y = \bigvee_{1 \leq u \leq v} \left( y_u \land r_{u+(n-v)} \land \left( \bigwedge_{1 \leq k < u} r_{k+(n-v)} \right) \right) = \bigvee_{1 \leq u \leq v} \left( y_u \land s_{v+1-u} \land \left( \bigwedge_{1 \leq k < u} s_{v+1-k} \right) \right).$$

Here, we let the output for $Y$ be $b_Y$ and let the output for $X$ be $b_X$. We omit some bits in the above formula because we mask the upper bits of the binary counter with NDROs and we pad 0s for unused bits of $XI$. We can represent the output of the AND gate as follows with

\[ \text{Algorithm 1 Flexible-precision multiplication with the proposed multiplier.} \]

1: Feed a pulse for $reset_{XY}, tr_1, \ldots, tr_f$ 
   // The values of operands are reset
2: Feed a pulse for $ts_1, \ldots, ts_{n-v}$ and feed operands $X = [0, x_1 \ldots x_v]_2$ and $Y = [0, y_1 \ldots y_v]_2$ through $XI$ and $YI$, respectively
   \( (x_1, \ldots, x_v, 0, \ldots, 0) \)
   \( (y_1, \ldots, y_{n-v+1}, \ldots, y_n) \)
3: Feed $2^v$ pulses for $clock$
4: Feed a pulse for $reset_{counter}$ and obtain the result from $Z$.

$[0, z_v z_{v-1} \cdots z_1]_2$ is the result
intermediate variables $P_{t,u}$:

$$P_{t,u} = \left\{ x_t \land y_u \right\} \land \left\{ \left( \bigwedge_{1 \leq k < t} s_k \right) \land \left( \bigwedge_{v+1-u < k \leq v} s_k \right) \right\}$$

$$b_X \land b_Y = \bigvee_{2 \leq t + u \leq v + 1} P_{t,u}.$$

The output of the AND gate $b_X \land b_Y$ is logic OR of $P_{t,u}$ for each pair of $x_t$ and $y_u$ ($2 \leq t + u \leq v + 1$). A value of $P_{t,u}$ is determined according to the output vector of the binary counter $(s_n, \ldots, s_1)$ and the values of $x_t$ and $y_u$. $P_{t,u}$ takes logic-1 when both $x_t$ and $y_u$ are logic-1 and the output vector of the binary counter is as follows:

$$(s_n, \ldots, s_1) = \begin{cases} (s, \ldots, s, 0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0) & \text{(if } 2 \leq t + u \leq v) \\ (s, \ldots, s, 0, \ldots, 0, 1, \ldots, 0) & \text{(if } t + u = v + 1) \end{cases} \tag{1}$$

where “*” denotes “don’t care”. When the binary counter feeds $2^v$ vectors, $(s_n, \ldots, s_1)$ takes all $v$-bit vectors from $(0, \ldots, 0)$ to $(1, \ldots, 1)$ during the period regardless of the starting state of the binary counter. When both $x_t$ and $y_u$ ($2 \leq t + u \leq v$) are logic-1 and the binary counter feeds $2^v$ vectors, there are $2^{v-(t+u)}$ vectors that make $P_{t,u}$ logic-1 because there are $v-(t+u)$ don’t cares. In $2^v$ vectors, there is one vector that makes $P_{t,u}$ logic-1 ($t + u = v + 1$) when both $x_t$ and $y_u$ are logic-1. Therefore, when the binary counter feeds $2^v$ vectors, the number of pulses the AND gate outputs is represented as $\sum_{t+u \leq v} x_t y_u 2^{v-(t+u)} + \sum_{t+u = v+1} x_t y_u$. We consider the weight of each pulse as $1 ulp$, i.e., $2^{-v}$.

The obtained result is $\sum_{t+u \leq v} x_t y_u 2^{v-(t+u)} + \sum_{t+u = v+1} x_t y_u 2^{-v}$.

### 3.3. Flexible-Precision Matrix Multiplication Circuit

We propose a flexible-precision matrix multiplication circuit with the proposed multiplier. We consider matrix multiplication $C = AB$ where each of $A$, $B$, and $C$ is an $m \times m$ matrix as follows:

$$\begin{pmatrix} C_{0,0} & \cdots & C_{0,m-1} \\ \vdots & \ddots & \vdots \\ C_{m-1,0} & \cdots & C_{m-1,m-1} \end{pmatrix} = \begin{pmatrix} A_{0,0} & \cdots & A_{0,m-1} \\ \vdots & \ddots & \vdots \\ A_{m-1,0} & \cdots & A_{m-1,m-1} \end{pmatrix} \begin{pmatrix} B_{0,0} & \cdots & B_{0,m-1} \\ \vdots & \ddots & \vdots \\ B_{m-1,0} & \cdots & B_{m-1,m-1} \end{pmatrix}.$$ 

Each element of input matrices is a $v$-bit fixed-point number ($v_{\text{min}} \leq v \leq n$). We represent $A_{i,j}$ as $[0.a_{i,j}^1 \cdots a_{i,j}^v]_2$. We show the structure of the circuit in Fig. 5. In the figure, we omit several control signals such as clock and reset signals. $BI$ and $AI_k(0 \leq k < m)$ are $n$-bit inputs $(b_1, \ldots, b_n)$ and $(a_{i,j}^1, \ldots, a_{i,j}^n)$, respectively. We extend the bit-width of pulse counters to $(n + l)$-bit where $l = \lceil \log_2(m) \rceil$. $CO_k(0 \leq k < m)$ are $(n + l)$-bit outputs $[c_{i,j}^k \cdots c_{i,j}^l]_2$ of the pulse counters. The circuit calculates a column of the resultant matrix by carrying out the following calculation:

$$\begin{pmatrix} C_{0,i} \\ \vdots \\ C_{m-1,i} \end{pmatrix} = \begin{pmatrix} A_{0,0} \\ \vdots \\ A_{m-1,0} \end{pmatrix} B_{0,i} + \cdots + \begin{pmatrix} A_{0,m-1} \\ \vdots \\ A_{m-1,m-1} \end{pmatrix} B_{m-1,i}. \tag{2}$$

For each $B_{0,i}$ ($0 \leq k < m$), the circuit performs $m$ multiplications of each term simultaneously. We realize the accumulation of multiplication results with the pulse counters by counting pulses.
4. Layout Designs and Evaluation Results

We have designed a layout of the flexible-precision multiplier and a layout of the matrix multiplication circuit for evaluating purposes. Both the designed layouts perform 3-bit and 4-bit multiplication. The designed matrix multiplication circuit was for $4 \times 4$ matrices, i.e., $m = 4$. We used the cell library for the AIST ADP2 process [13]. We show the designed layouts in Figs. 6 and 7. We tuned delay time in them for 40GHz operation.

For the layouts, we used a design in Fig. 8 for the binary counter. RTFFB is a resettable toggle flip-flop (TFF) with both inverted and non-inverted outputs. Each RTFFB outputs a pulse every cycle at one of two output terminals alternately. We use those two outputs $dout0$ and $dout1$ as the sum signal and the carry signal, respectively. The each of NDROs is set by a pulse on the sum signal, i.e., $dout0$ and is reset by the carry signal, i.e., $dout1$.

The number of Josephson junctions (JJs) in the multiplier is 1,049 and its area is 0.43 mm$^2$ ($0.51 \times 0.84$ mm). The number of JJs in the matrix multiplication circuit is 2,308 and its area is $84$ mm$^2$. The number of Josephson junctions (JJs) in the multiplier is 1,049 and its area is 0.43 mm$^2$ ($0.51 \times 0.84$ mm). The number of JJs in the matrix multiplication circuit is 2,308 and its area is $84$ mm$^2$.

![Diagram](image_url)

**Figure 5.** Structure of the flexible-precision matrix multiplication circuit.

Algorithm 2 Flexible-precision matrix multiplication with the proposed circuit.

| Step | Description |
|------|-------------|
| 1:   | Feed a pulse for $tr_1, \ldots, tr_f$ |
| 2:   | Feed a pulse for $ts_1, \ldots, ts_{m-v}$ |
| 3:   | for $i = 0 \ldots m-1$ do |
| 4:   | for $k = 0 \ldots m-1$ do |
| 5:   | Reset the operand values of BI, AI$_0$, $\cdots$, AI$_{m-1}$ |
| 6:   | Feed operands to the circuit |
| 7:   | Feed 2$^v$ clock pulses |
| 8:   | end for |
| 9:   | Feed a pulse for $reset_{count}$ and read a column of the result |
| 10:  | end for |

We previously proposed a matrix multiplication circuit [12] based on the original bit-level multiplier in [7]. The proposed circuit can change the precision in matrix multiplication.
1.44 mm² (1.71 × 0.84 mm). We have verified the valid operation of logic level designs of those two circuits with the logic-level simulation tool [14] before designing the layouts, and verified their valid operation with Verilog netlists extracted from the layouts. There have been small bit-serial multipliers such as [9, 10]. The number of JJs in the 4-bit design [9] utilizing special cells was 1,097 and the number of JJs in the four PEs in [10] was 2,556. Though the figures depend on design style of layouts, the number of JJs in the proposed multiplier is smaller. When we implement plural proposed multipliers, they can share many component circuits. The number of JJs in the matrix multiplication circuit is smaller than the estimated number of JJs in the previously proposed circuit in [11].

The proposed multiplier performs truncated multiplication and its result contains small error. We evaluated the error of the proposed circuits. We show the result in Table 1. The maximum and average error were evaluated in ulp for the exhaustive input patterns. For example, the multiplier calculates \([0.0101]_2 \times [0.0111]_2\) as \([0.0010]_2\) while the true result is \([0.00100011]_2\). In this case, we considered the error in ulp as 0.1875 (= \([0.0011]_2\)) derived by \(\|[0.00100011]_2 - [0.0010]_2\|/ulp\). In the table, we also show the error for rounding the true value to the nearest \(v\)-bit fixed-point one and the error for rounding towards zero, i.e., cutting the lower bits. When we round to the nearest value or round towards zero, the maximum error is 0.5 ulp or \((1 - 2^{-v})\) ulp, respectively. The maximum error of the proposed multiplier increases from 0.9 ulp to 1.6 ulp depending on the bit-width because the number of partial products, i.e., the rows in Figure 1, increases. An example of the pair of \(X\) and \(Y\) for the maximum error in Table 1 is \(X = Y = [0.1010 \cdots 101]_2\) for odd \(v\) and \(X = [0.10110 \cdots 1011]_2\) and \(Y = [0.1010 \cdots 1011]_2\) for even \(v\). When it performs the 5-bit operation \([0.10101]_2 \times [0.10101]_2\),
Table 1. Maximum and average error of the proposed multiplier in ulp.

| bit width | Proposed multiplier | Round to the nearest | Round to zero |
|-----------|---------------------|----------------------|---------------|
|           | Maximum  | Average  | Maximum | Average  | Maximum  | Average  |
| 3         | 0.875    | 0.281    | 0.500   | 0.219    | 0.875    | 0.348    |
| 4         | 1.063    | 0.323    | 0.500   | 0.234    | 0.938    | 0.406    |
| 5         | 1.219    | 0.353    | 0.500   | 0.242    | 0.969    | 0.445    |
| 6         | 1.391    | 0.377    | 0.500   | 0.246    | 0.984    | 0.469    |
| 7         | 1.555    | 0.396    | 0.500   | 0.248    | 0.992    | 0.482    |

the error is $[1.00111]_2 (= 1.219)$ ulp. The maximum error is not very large up to around 5 bits compared with the maximum error of rounding towards zero. The average error of the multiplier is smaller than the average error of rounding towards zero for all bit-widths.

5. Conclusion

We proposed a flexible-precision multiplier for RSFQ circuits. It utilizes bit-level processing and is extended to perform flexible-precision multiplications from the previously proposed fixed-precision circuit. As the bit-width of the operation decreases, the latency in cycle decreases. We can achieve higher multiplication performance for processes tolerating larger error. We also show a flexible-precision matrix multiplication circuit utilizing the proposed multiplier. Because its internal multipliers share many component circuits, we can realize the circuit in a compact area. The number of JJs are small compared with previously proposed designs for RSFQ circuits. The proposed circuits are suitable for applications which tolerate small error and request many multiplications.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP19K11888 and supported by VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration with Cadence Corporation.

References

[1] Likharev K and Semenov V 1991 IEEE Transactions on Applied Superconductivity 1 pp 3–28
[2] Mrazek V, Sarwar S S, Sekanina L, Vasic Z and Roy K 2016 Design of power-efficient approximate multipliers for approximate artificial neural networks Proc. Int. Conf. Computer-Aided Design (ICCAD)
[3] Zhang H, Chen D and Ko S 2020 IEEE Transactions on Computers 69 pp 26–38
[4] Nannarelli A 2019 IEEE Transactions on Computers 68 pp 1553–1560
[5] Dorojevets M, Kasperek A K, Yoshikawa N and Fujimaki A 2013 IEEE Transactions on Applied Superconductivity 23 1300104
[6] Nagaoka I, Tanaka M, Inoue K and Fujimaki A 2019 IEEE International Solid-State Circuits Conference (ISSCC2019) pp 460–462
[7] Kito N, Odaka R and Takagi K 2019 IEICE Transactions on Electronics E102-C pp 607–611
[8] Alaghi A and Hayes J P 2013 ACM Transactions on Embedded Computing Systems 12 pp 92:1–92:19
[9] Herr Q P, Vukovic N, Mancini C A, Gaj K, Qing Ke, Adler V, Friedman E G, Krasniewski A, Bocko M F and Feldman M J 1997 IEEE Transactions on Applied Superconductivity 7 pp 3168–3171
[10] Hara H, Obata K, Park H, Yamanishi Y, Taketomi K, Yoshikawa N, Tanaka M, Fujimaki A, Takagi N, Takagi K and Nagasawa S 2009 IEEE Trans. Appl. Supercond. 19 pp 657–660
[11] Tang G, Qu P, Ye X, Fan D and Sun N 2018 IEEE Transactions on Applied Superconductivity 28 1300905
[12] Kito N, Kumagai T and Takagi K 2019 Proc. 22nd Workshop on Synthesis And System Integration of Mixed Information Technologies (SAMSI 2019) pp 99–103
[13] Nagasawa S, Hinode K, Satoh T, Hidaka M, Akaike H, Fujimaki A, Yoshikawa N, Takagi K and Takagi N 2014 IEICE Transactions on Electronics E97-C pp 132–140
[14] Kito N, Udatsu S and Takagi K 2020 Journal of Physics: Conference Series 1590 012041