Temporal and SFQ Pulse-Streams Encoding for Area-Efficient Superconducting Accelerators

Patricia Gonzalez-Guerrero
lg4er@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Meriam Gay Bautista
mgbautista@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Darren Lyles
dlyles@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

George Michelogiannakis
mihelog@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

ABSTRACT

Superconducting technology is a prime candidate for the future of computing. However, current superconducting prototypes are limited to small-scale examples due to stringent area constraints and complex architectures inspired from voltage-level encoding in CMOS; this is at odds with the *ps*-wide Single Quantum Flux (SFQ) pulses used in superconductors to carry information. In this work, we propose a wave-pipelined Unary SFQ (U-SFQ) architecture that leverages the advantages of two data representations: pulse-streams and Race Logic (RL). We introduce novel building blocks such as multipliers, adders, and memory cells, which leverage the natural properties of SFQ pulses to mitigate area constraints. We then design and simulate three popular hardware accelerators: i) a Processing Element (PE), typically used in spatial architectures; ii) A dot-product-unit (DPU), one of the most popular accelerators in artificial neural networks and digital signal processing (DSP); and iii) A Finite Impulse Response (FIR) filter, a popular and computationally demanding DSP accelerator. The proposed U-SFQ building blocks require up to 200× fewer JJs compared to their SFQ binary counterparts, exposing an area-delay trade-off. This work mitigates the stringent area constraints of superconducting technology.

CCS CONCEPTS

• **Hardware → Emerging technologies**; • **Theory of computation → Modal and temporal logics**; • **Computer systems organization → Architectures**.

KEYWORDS

superconducting logic; temporal logic; race logic; pulse-streams arithmetic; finite impulse response; dot product unit; digital signal processing

ACM Reference Format:
Patricia Gonzalez-Guerrero, Meriam Gay Bautista, Darren Lyles, and George Michelogiannakis. 2022. Temporal and SFQ Pulse-Streams Encoding for Area-Efficient Superconducting Accelerators. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28 – March 4, 2022, Lausanne, Switzerland*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3503222.3507765

ASPLOS '22, February 28 – March 4, 2022, Lausanne, Switzerland

ACM ISBN 978-1-4503-9205-1/22/02.
https://doi.org/10.1145/3503222.3507765

1 INTRODUCTION

Metals have the reputation of being good conductors. However, certain metals exhibit zero electric resistance at low temperatures, making them *superconductors*! Superconductivity is the property of certain metals to have zero resistance below a critical temperature that is usually a few degrees Kelvin [50]. While in the semiconductor realm, the MOSFET and the CMOS logic family have dominated digital design, in superconductors, the Josephson junction (JJ), an ultra-fast switching device, and the Rapid-Single-Flux-Quantum (RSFQ) logic family [30] have been the most popular approach. In RSFQ, a logic transition is represented as pico-seconds-wide, tens-of-millivolts-amplitude pulse (Figure 1) which enables processing speeds of tens of GHz [26, 28, 50] while keeping the switching energy six orders of magnitude less than current CMOS processes [33, 50].

Typical superconducting architectures implement logic using CMOS-inspired data representation, predominantly with AND-OR logic. Because in RSFQ AND-OR gates are synchronous, RSFQ logic circuits today often use deeply-pipelined datapaths where almost every cell in the design must be synchronized with a global clock [30, 51]. These deeply-pipelined structures suffer from control and data hazards and stringent timing constraints that result in expensive clock trees [21]. Besides architectural challenges, manufacturing limitations restrict the number of active devices to just a few tens of thousands of JJs per die [5]. Thus, superconducting circuits have been limited to relatively low scales.

Alternative computing paradigms might hold the key to compute in a language that is more natural to SFQ. Unary computing is a promising solution to improve the area and energy efficiency of massively-parallel computing [55]. In CMOS unary computing, a number is represented as a stream of high and low voltages, enabling computing with a very low area footprint. Some typical examples of CMOS unary computing are Synchronous Stochastic Computing (SSC) [14] and Race Logic (RL) [29]. Other more exotic flavours include asynchronous stochastic computing [15] and computing with ΣΔ-Modulated [16] or Pulse-Width-Modulated

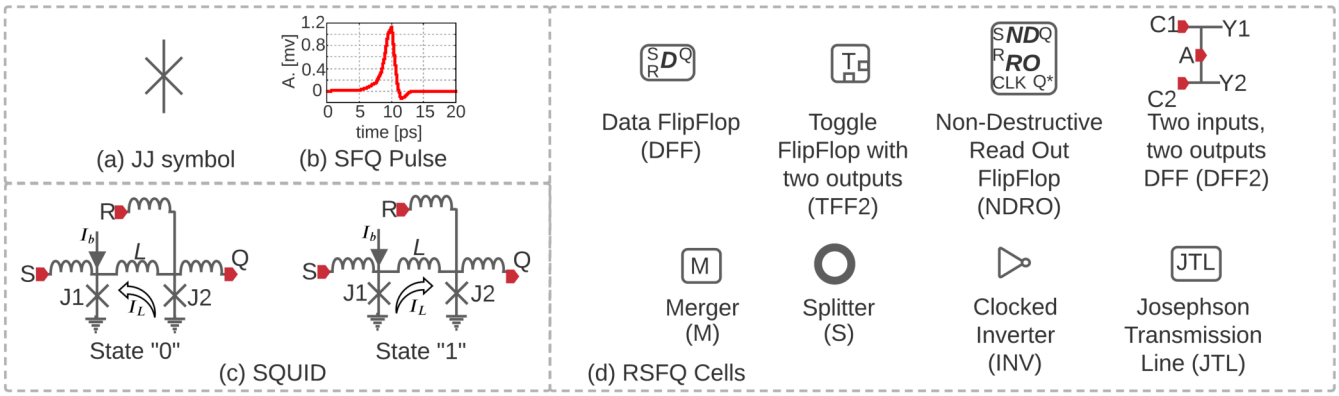

Figure 1: SFQ fundamentals. (a) The JJ is an ultra-fast switching device built as a superconductor-insulator-superconductor sandwich. (b) Data is represented as ps wide SFQ pulse. (c) The SQUID is the basic building component in the RSFQ logic family. (d) With the SQUID we can build the basic SFQ gates such as the DFF, TFF2, NDRO, DFF2, and inverter. The merger, splitter, and JTL are stateless cells used to facilitate interconnection between gates [11, 58].

streams [39]. A typical CMOS SSC multiplier is implemented as a simple AND gate [14]. Calculating the minimum between two values in RL requires a single OR gate [29], yielding area savings of more than 90% compared to calculating the minimum of two binary values.

Previous work has explored unary approaches such as SSC and RL in SFQ technology, demonstrating beneficial trade-offs between area and energy/power consumption for selected applications [6, 51]. Stream-based methods, such as SSC, are particularly efficient for mathematical operations yet sacrifice accuracy to latency. On the other hand, RL yields area efficiency for dynamic programming algorithms [29], yet it is expensive for arithmetic computations such as multiplication and addition. *Can we leverage the advantages of both pulse-streams and RL?*

This paper addresses RSFQ architectures challenges by proposing a novel wave-pipelined Unary SFQ (U-SFQ) architecture. Our work combines the best of computing with pulse streams and RL, leveraging SFQ pulses as an efficient form of data representation. Our contributions are as follows:

- **Data representation:** Section 3 extends RL to make it suitable for arithmetic operations and defines SFQ pulse stream characteristics for efficient computing.
- **U-SFQ Building Blocks:** Section 4 introduces novel building blocks such as multipliers, adders, and memory elements for unary SFQ data representation.
- **Architecture Evaluation:** We evaluate the potential of the proposed U-SFQ architecture in Section 5 by analyzing three hardware accelerators. First we introduce a PE for Coarse-Grained Reconfigurable Architectures (CGRAs) or Spatial Architectures (SpA) for CNNs. Second, we show a dot-product unit (DPU), which is a basic building block of artificial neural networks and Digital Signal Processing (DSP). Finally, we evaluate a novel programmable Finite Impulse Response (FIR) filter accelerator. FIR filters are one of the most used DSP hardware accelerators but also demand a high number of hardware resources given the high number of additions and multiplications [53, 56].

Our proposed U-SFQ processing elements yield up to 200× savings in area (number of JJs) compared to their binary counterparts. This results in 28%-98% savings in area for a U-SFQ PE array compared with its binary counterpart for the same throughput. Moreover, our U-SFQ architecture is resilient to computing errors. In binary, a 30% error rate results in +30dB of signal to noise ratio (SNR) degradation, versus only 4dB for our U-SFQ architecture.

2 BACKGROUND AND MOTIVATION

2.1 RSFQ Logic

Superconductors are metals with zero resistance below critical temperatures [50]. The warhorse for superconducting circuits is the JJ [45] (Figure 1a) that allows current through with no resistance until a critical current I_c. Reaching I_c causes the JJ to switch to a resistive state and emit a *ps* wide voltage pulse [30].

A basic building block for RSFQ logic is the Superconducting QUantum Interference Device (SQUID). The SQUID, shown in Figure 1c, is built using two JJs (J_1,J_2) connected by an inductance L, and has two stable stationary states that differ by the direction of the persistent current I_L circulating in the loop. Assume that I_L circulates counterclockwise so that the current in $J1 = I_b/2 + I_L \approx 0.7I_c$, with I_b the bias current. This state represents a zero. If an SFQ pulse arrives at input S, the current through J_1 reaches its critical current, provoking a quick superconducting→resistive→superconductive transition (also known as a kickback), and the direction of I_L becomes clockwise. A clockwise current direction represents a "1". Now, if an SFQ pulse occurs at input port R, the current through J_2 reaches its critical current and the cell reverts to a state "0". This fast kickback generates an SFQ pulse at J2, which will be propagated at output Q [34].

In RSFQ logic, the SQUID is the basis of all traditional binary logic gates [58]. Table 1 describes the RSFQ gates of interest for this work while Figure 1d shows the cells symbols.

Table 1: Relevant superconducting gates [11, 58]. *Acronym

Acro*	Summary
S	Produces a pulse at both outputs per input pulse.
M	Produces a pulse at the output for a pulse at either input.
JTL	Acts as a buffer, sharpening the output pulse.
FA	Produces an output pulse at the first time an input pulse arrives at either of its two inputs.
DFF	S sets the SQUID to 1. R resets the SQUID and generates an output pulse.
DFF2	A sets the SQUID to 1. C1(C2) resets the SQUID and generates an output pulse at Y1(Y2).
TFF2	Distributes the incoming pulses through alternating output ports.
NDRO	Ports S, R, and Q resemble a DFF. A pulse at the CLK port reads the SQUID's state without altering it.

2.1.1 Binary Multipliers and Adders in RSFQ. Two major architectures dominate the design of multipliers and adders in SFQ technology: a) bit parallel and b) wave-pipelined. In bit-parallel architectures, every cell is clocked. The frequency of the pipeline is dictated by the maximum cell propagation time added to the cell's setup and hold times. Recent work demonstrated a 48GHz 8 bits multiplier, which is 48GOPs of throughput, using 17K JJs [37]. Typically in bit-parallel architectures, there is a significant synchronization overhead [8].

In wave-pipelining, a calculation starts as soon as both operands arrive and advances as a flow of pulses instead of waiting for the clock. Wave-pipelining yields up to 3× better area metrics due to the absence of a clock signal per cell, yet it has a performance penalty due to the extra time required to avoid data collisions among the data waves [9, 10, 12]. Table 2 summarizes key metrics for representative adders and multipliers from literature. Throughout the paper, we compare our proposed adders and multipliers with the ones summarized in this table.

Table 2: State of the art for RSFQ multipliers and adders. BP=Bit-Pipelined, WP=Wave-Pipelined, K=Kogge-Stone, S=Sparse-Tree, NG=Northtrop Grumman, SA=Systolic Array, C=Carry-Save. *Projected from [8]. ** Projected from [40]

Ref.	Bits	JJ count	Latency (ps)	Arch.	Technology
Adder					
[23]	4	931	50	BP	KOPTI $1.0kA/cm^2$Nb
[41]	8	6581	588	WP, K	AIST-STP2
[8]*	8	4351	222	WP, K	NG
[8]	16	16683	255	WP, K	NG
[9]	16	9941	352	WP, S	ISTEC$1.0\mu m10kA/cm^2$
Multiplier					
[40]	4	2308	1250	SA	NEC $2.5kA/cm^2$
[40]	8	4616	2540	SA	**
[37]	8	17000	333	BP	$1\mu m$Nb/AlOx/Nb
[10]	8	5948	447	WP, C	ISTEC$1.0\mu m10kA/cm^2$
[40]	16	9232	5120	SA	**

2.1.2 SFQ Power Considerations. SFQ power consumption can be divided in three major components: i) *active* due to switching, which is proportional to the number of JJs and the activity factor; ii) *passive* due to biasing of the JJs; and iii) cooling power. Although RSFQ is the pioneer and most popular logic family, it suffers from high passive power consumption due to using resistors for biasing which results in a constant current draw. To address this, logic families such as energy-efficient RSFQ (ERSFQ) [33] replace biasing resistors for a network of JJs eliminating the passive power consumption at the expense of a slight increment in the area. With ERSFQ, the performance per W of superconducting designs can be 493×-1.23× higher than a CMOS implementation without and with the cooling costs respectively [21]. Moreover, we believe that ongoing work in cryocooling fostered by the advancement of cryosensors and quantum computing will further increase the energy and power savings of SFQ compared with CMOS [7]. Given that previous work has demonstrated the potential for superconducting circuits for better throughput and energy efficiency [21, 38, 50], in this work we focus on comparing binary SFQ architectures with our proposed U-SFQ architecture.

2.1.3 SFQ Roadmap and Challenges. Superconducting technology has the potential to address the current challenges of high performance computing. However, the following limitations hinder wider-spread adoption of superconducting digital computing:

(1) **Limited device density:** There is a 1000× device density gap between superconducting technology and CMOS. This limited device density is exacerbated by the fact that SFQ logic requires more devices to implement traditional binary designs than CMOS [50].
(2) **Reliability:** JJs are susceptible to flux-trapping and manufacturing defects. This results in a fault mode that is significantly different than CMOS. Architecture-level solutions such as redundancy have been proposed [50], exacerbating the device density limitations mentioned before. Another option is the development of appropriate EDA tools [13] that include this fault modes in the synthesis and P&R flow.
(3) **Memory:** Although cryogenic memories such as (i) Vortex-Transition-Memory (VTM), (ii) Josephson-CMOS SRAM, (iii) Magnetic RAM (MRAM), and (iv) superconducting nanowire RAM (SNM) have been proposed [49, 52], there are still density, integration and performance limitation to be addressed. For practical reasons the most viable on-chip memory for SFQ technologies uses DFF-based shift registers [21].
(4) **EDA tools:** have been developed as part of academic research efforts mostly focused on device and gate level simple synthesis that reaches thousands of gates [27]. To achieve large scale integration capable of reaching billions of active devices, tools that enable back-annotation from the layout to schematic, and parameter extraction are still required [13]. Programs such as Supertools [20] are tackling the EDA challenge.

2.2 Alternative Data Representations

2.2.1 Race Logic. Recent work explored Race Logic (RL) as an alternative way of encoding and processing information using RSFQ

logic [51]. In RL, information is encoded as a delay from a reference signal. Computation is performed by observing the relative propagation times of signals injected into a circuit (*i.e.* the outcome of races) [29]. For example, Figure 2a shows the computation of the minimum between two RL signals. FA requires only 8 JJs [51], while a binary implementation of the minimum between two numbers requires more than 4K JJs. Because RL encodes data in time, it is well-suited for dynamic programming [29] but basic arithmetic operations such as addition and multiplication are expensive.

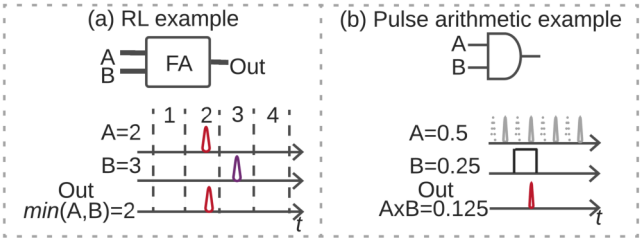

Figure 2: (a) Example of computing the minimum between two RL signals A=2 and B=3 with the FA primitive [51]. (b) Example of CMOS pulse stream multiplication. A=0.5 is a pulse stream, while B is a CMOS signal that is high 25% of the time. For this example, $P_{max} = 8$ and the output represents $1/8 = 0.125$ [35]

2.2.2 CMOS Pulse Stream Arithmetic. "pulse stream" encoding in biological networks might hold the key to mitigate RL limitations. When a neuron is ON, it fires a regular train of voltage spikes. When the neuron is OFF, it does not. This biologically-inspired, impulse-based data representation kick-started pulse stream arithmetic and is mostly used in spiking neural networks [35, 36]. In pulse stream arithmetic, a number p is mapped to the rate (R_p) of a train of voltage spikes such that $R_p = 0$ for $p = 0$, and $R_p = R_{max}$ for $p = 1$. Consequently, there is a maximum number of pulses N_{max} that occurs when $p = 1$. Then, p is recovered from the pulse stream by counting the number of pulses and dividing them by N_{max}.

Opposite to RL, pulse stream representation favours mathematical operations such as multiplication and addition. For example, multiplication is performed by filtering out a fraction of the pulses with an AND gate [36]. Figure 2b shows an example of pulse stream multiplication with A=0.5 represented as a stream of pulses firing at half the maximum frequency and B=0.25 represented as a CMOS level signal. The result y can be obtained by $y = N_y/N_{max}$ where N_y is the number of pulses coming out of the AND gate.

3 UNARY SFQ DATA REPRESENTATION

Inspired by the natural occurrence of pulses in SFQ logic, we combine pulse stream-arithmetic with RL to propose a *unary SFQ* data representation. We aim to alleviate SFQ area constraints by designing processing elements with a minimum area footprint while leveraging the high SFQ processing speeds.

3.1 Race Logic for Mathematical Operations

To understand the proposed U-SFQ data representation, let us start from RL. Figure 3a shows a typical RL epoch. We define an epoch to be subdivided into time slots. To represent number 3, the SFQ pulse arrives at the time slot identified as Id=3. We then modify the original RL definition by dividing the time slot Id by the maximum time slot in the epoch, thus obtaining a numerical representation between 0 and 1. This unipolar data representation enables operations for positive numbers, which agrees with the natural evolution of time. Inspired by the bipolar stochastic computing representation [14], we can obtain a RL bipolar representation by scaling and shifting the unipolar Id such that $Id_b = 2Id_u - 1$, where Id_u is the unipolar Id and Id_b the equivalent bipolar Id.

3.2 SFQ Pulse Train Data Representation

We propose to map a number p to the frequency R_p of a periodic train of SFQ pulses, inspired by pulse stream arithmetic (Section 2.2.2). A unipolar data representation maps the maximum rate R_{max}, or equivalently the maximum number of pulses N_{max} in an epoch, to 1 and the absence of pulses to 0. Notice that $p = n/N_{max}$, where n is the number of pulses during a computing epoch. Also, each pulse has an associated weight of $1/N_{max}$. Similar to SSC [14], to enable negative numbers we can derive a bipolar data representation as $p_b = 2p_u - 1$, where p_u is the unipolar representation and p_b is the equivalent in a bipolar representation.

4 UNARY SFQ BUILDING BLOCKS

Because direct mapping of CMOS-pulse stream-arithmetic building blocks to SFQ is expensive, we can combine pulse streams with the compactness of RL to enable efficient processing units. In this section, we introduce our novel unary SFQ multiplier and unary SFQ adder.

4.1 Unary SFQ Multiplier

Given two numbers p_A and p_B in the range of $[0, 1]$, we propose to encode p_A as the rate of an SFQ pulse stream A, and p_B as a RL signal B. To calculate $p_A \times p_B$, we use the RL signal B to filter out a percentage of pulses from A (see Figure 3b). What remains after the filtering process, encoded in a pulse stream, is the multiplication result. To implement this, we use an NDRO cell as shown in Figure 3c (left).

Using a single NDRO, we can multiply positive numbers. That is called unipolar multiplication. To multiply negative numbers, we propose a bipolar multiplier shown in Figure 3c (right). Notice that the proposed multiplier is inspired by a CMOS XNOR, that is the bipolar multiplier for stochastic computing [14]. Before the arrival of the RL pulse B, the top NDRO lets the pulse stream A pass through. As a result, $O1 = A \wedge B$. When B arrives, it sets the bottom NDRO letting $\neg A$ pass. As a result, $O2 = \neg A \wedge \neg B$. The merger combines $O1$ and $O2$ obtaining $OUT = (A \wedge B) \vee (\neg A \wedge \neg B)$.

To evaluate the performance of our proposed multiplier, we design and simulate the circuit with WRSpice using the open-source *MIT-LL SFQ5ee* $10kA/cm^2$ process. Figure 4 shows the latency and area of unary and binary multipliers.

Area: The area (number of JJs) of the binary multiplier is proportional to the number of bits, while the area of the unary multiplier remains constant. Our proposed multiplier occupies 25× to 200× less area than the binary Wave-Pipelined (WP) architecture. When

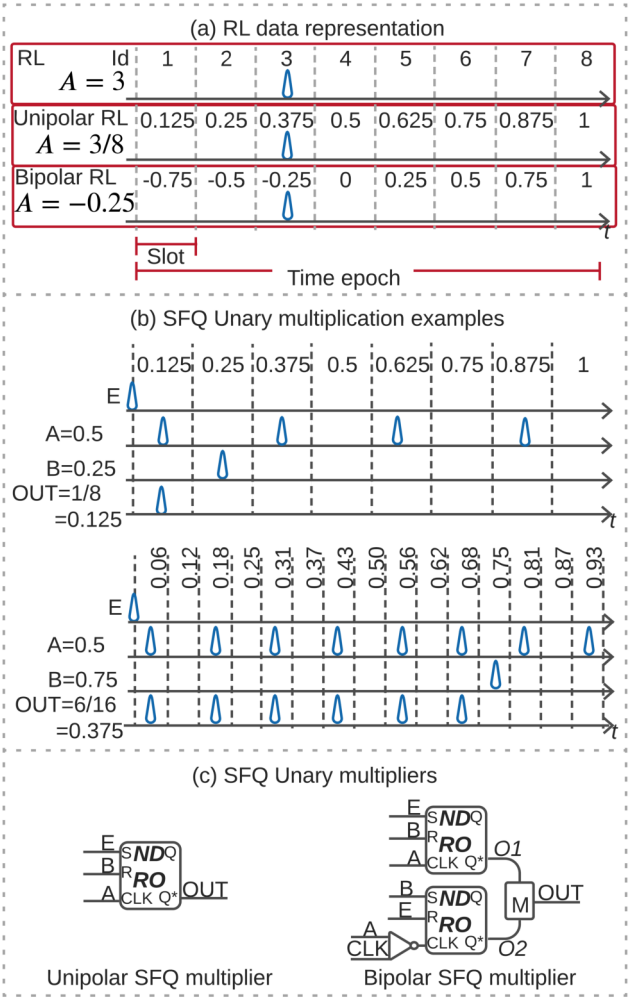

Figure 3: (a) Conventional RL data representation and our proposed unipolar and bipolar RL data encoding. (b) Unipolar multiplication examples. E signals the beginning of the epoch. The first example with a 3-bit resolution ($N_{max} = 8$) results in $0.125 = 1/N_{max}$. The second example with a 4-bit resolution ($N_{max} = 16$) results in $0.375 = 6/N_{max}$. (c) Proposed unipolar and bipolar multipliers. *Unipolar multiplier*: E sets the SFQ loop to "1", while RL input B sets the SFQ loop to "0". pulse stream A is connected to the non-destructive-read input port CLK. Pulses arriving before B pass through the NDRO while pulses after B do *not*. *Bipolar multiplier:* the top NDRO behaves as the unipolar version, while the bottom NDRO is set at the arrival of the RL input and resets at the beginning of the epoch.

compared with the BP binary architecture in [37], the unary multiplier yields 370× savings in area.

Latency: The simulated delay for our proposed multiplier is $t_{INV} = 9ps$, which corresponds to the propagation, setup and hold time for the inverter. This, also results in maximum frequency of

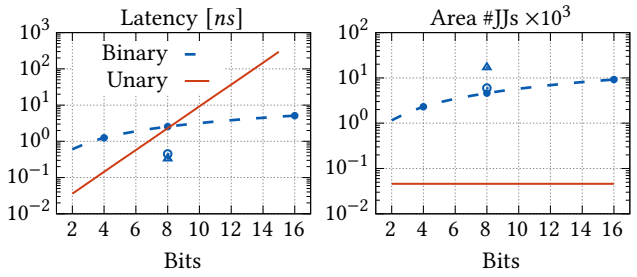

Figure 4: Latency and area comparison between the proposed U-SFQ multiplier and binary multipliers. △ [37], ○ [10], • [40], (Table 2). The dashed line is a linear fit of the data reported in Table 2.

$\approx 111GHz$, which translates to a computation latency of $2^B t_{INV}$ with B the bit resolution. The latency of the binary multiplier is linearly proportional to B, while the latency of the unary multiplier increases exponentially with B. Compared with a BP multiplier [37], the binary architecture is 6× faster than U-SFQ at the expense of 370× more area for 8 bits. When compared with the WP architecture, the unary multiplier is faster for less than 8 bits.

4.2 Unary SFQ Addition

This section discusses two possible implementations for a unary SFQ adder: (A) a merger and (B) a counting network.

(A) Unary SFQ Addition With a Merger: One method for addition merges pulses into a single stream. A typical 2:1 SFQ merger cell has two inputs (A, B), one output (Y), and resembles the behavior of a CMOS OR gate (Figure 5a). The addition result $p_Y = p_A + p_B$ is obtained by counting the number of pulses at the merger's output and dividing by the maximum number of pulses expected in the computation time.

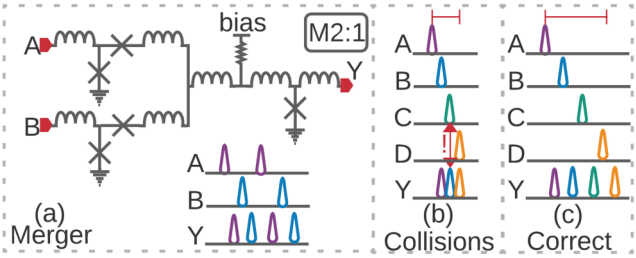

Figure 5: (a) Typical SFQ merger built with 5 JJs [58] and its operation. (b) Pulse collision in a 4:1 merger cell. (c) Correct operation for a 4:1 merger cell with a latency increase.

Computation error arises from the arrival of two pulses simultaneously at the merger input ports. In this case, only one pulse appears at the output Y. Therefore, the architecture must guarantee that pulses do not collide. This increases the minimum pulse spacing, increasing the computation latency. Moreover, the distance between input pulses is dictated by the intrinsic delay of the merger cell. Figure 5b shows an error example for a 4:1 merger cell (an M:1 merger cell can be built from 2:1 merger cells). Notice that four

pulses come in and only three come out. Figure 5c shows how to avoid collisions by increasing computation latency. The minimum distance between pulses increases with the number of inputs.

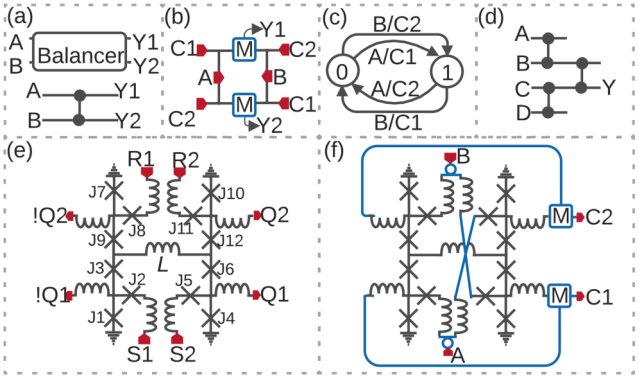

Figure 6: (a) 2:2 balancer. (b) Balancer output stage. (c) Mealey machine of the balancer routing logic. (d) 4:1 counting network. (e) B-Flip Flop (BFF). (f) Balancer routing unit.

(B) Unary SFQ Addition With a Counting Network: To avoid delay injection to avoid collisions, we propose to use an M:1 counting network (M inputs, one output) where M is a power of two [4, 6, 31]. A balancer (Figure 6a) is the building block for counting networks. A balancer has two inputs and two outputs, and acts like a toggle mechanism that repeatedly sends one incoming pulse to the top output and one to the bottom, balancing the number of pulses at both outputs [4]. The advantage of the balancer over the merger is its ability to deal with collisions. When pulses appear simultaneously at inputs A and B, a pulse is generated at each output. Consequently, each balancer's output generates $(N_A+N_B)/2$ pulses, with N_A, N_B the number of pulses at input A and B, respectively. The addition $p_Y = (p_A + p_B)/2$ is obtained by counting the number of pulses at either Y1 or Y2 and dividing by N_{max}.

We use an M:1 counting network to accumulate pulses coming from M parallel inputs. Figure 6d shows an example of a 4:1 counting network built with three balancers. Since the balancer divides the number of input pulses by two, the total number of pulses at the counting network output is $(N_{A1} + N_{A2} + ... + N_{AM})/M$, with $N_{A1}, N_{A2}, ..., N_{AM}$ the number of pulses at the $A1$, $A2$,..., AM input ports. The addition result $p_Y = (p_{A1} + p_{A2} + ... + p_{AM})/M$ is the number of pulses at the counting network output divided by N_{max}.

A balancer can be implemented with a merger followed by a TFF2 [31]. However, when two pulses arrive simultaneously, one is lost to the collision. Therefore, we propose a novel RSFQ balancer for pulse streams that is composed of two circuits: (i) an output stage and (ii) a routing unit.

(i) Balancer Output Stage: To enable the propagation of two simultaneous incoming pulses, we design the output stage shown in Figure 6b. The stage is composed of two DFF2s facing each other through merger cells. The SFQ loop for the DFF2 on the right (left) is set to "1" by input A (B). After this, control signal C1 (C2) reads the DFF2 state through the top (bottom) output. C1 and C2 are generated by the routing unit as described below.

(ii) Balancer Routing Unit: To generate C1 and C2, we design a routing unit following the Mealy machine shown in Figure 6c. An incoming pulse from either A or B causes a toggle between states 0 and 1. Outputs C1 and C2 depend on both the state and the inputs. Figure 6f shows the routing unit which is based on the B-Flip Flop (BFF) [43]. The BFF (Figure 6e) has four inputs that modify a single quantizing loop with two stationary states. The quantizing loop is formed by an inductance L closed via the ground and two 4-JJ-low-inductance-loops [58]. We connect the balancer's input A (B) to S1 and R2 (S2 and R1) through splitter cells. Then we merge Q1 and !Q1 (Q2 and !Q2) to generate outputs C1 (C2).

Figure 7 shows a waveform of our proposed balancer. The first pulse at input port B causes the state to transition from "0" to "1" (green), generating a pulse at output Y1. The next incoming pulse at either input port A or B will provoke a pulse at output Y2 and the state reset. When pulses arrive at inputs A and B simultaneously ($\sim 7ps$), there is one pulse at *each* output. Each output produces a number of pulses equal to the total number of pulses coming from A plus B divided by 2.

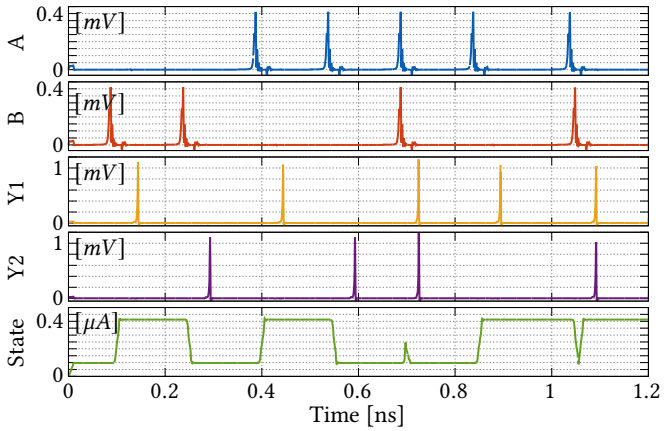

Figure 7: Balancer waveforms. A and B are the inputs, while Y1 and Y2 are the outputs. The last plot is the BFF state.

For the BFF state sequence, three different cases can be observed: (i) Pulses at A and B arrive at different times; (ii) Pulses at A and B arrive at the same time; and (iii) A second pulse arrives when the BFF state is transitioning. For cases (i) and (ii), the behavior of the cell follows the Mealy machine in Figure 6c. Moreover, because the output stage handles two pulses simultaneously, no additional action is needed.

Case (iii) causes unwanted behavior. If a second pulse arrives at either input during BFF transition ($t_{BFF} = 12ps$), this pulse is ignored by the control logic. Although the output is still correct because the output unit still generates two pulses, over time the balancer might be biased towards one output. We discuss the impact of these errors in computation accuracy in Section 5.4.1. To avoid this situation, we must guarantee a minimum delay of t_{BFF} between input pulses. Thus, the adder latency is set by $t = 2^B t_{BFF}$ with B the bits resolution.

Figure 8 compares the latency and area for the 2:1 merger unit, balancer, and binary adder. We compare our adders with the adders reported in literature (Table 2). Both the merger and balancer yield

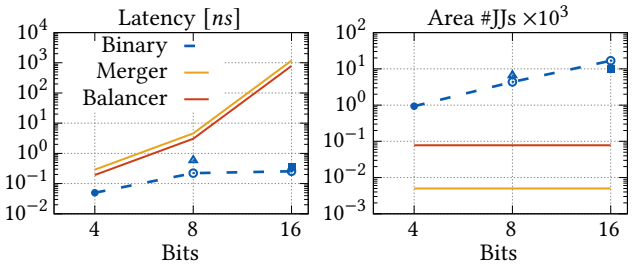

Figure 8: A comparison of the unary SFQ and binary adders. △ [41], ○ [8], • [23], . [9], (Table 2). Dashed-line only for visualization.

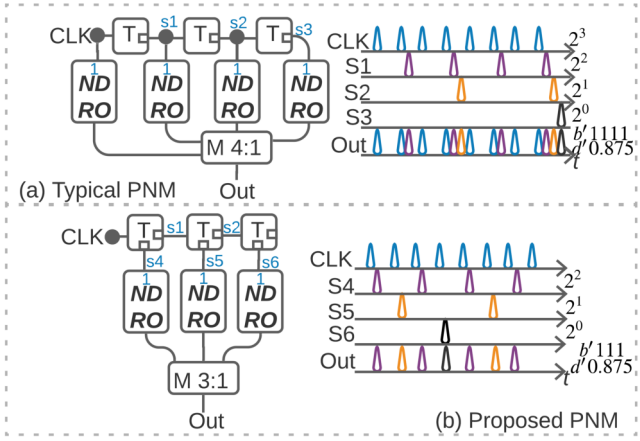

Figure 9: a) Typical Pulse Number Multiplier (PNM) implemented with TFFs [32, 46, 48]. b) Proposed PNM using TFF2s to generate a stream resembling a uniform rate.

area savings compared to the binary adder, but with a latency penalty. The balancer yields 11×-200× area savings versus the binary adder for 4-16 bits.

4.3 Memory Bank for Coefficients

In typical DSP and neural network accelerators, inputs are multiplied with coefficients or weights. Since these coefficients are loaded in memory once and rarely get updated thereafter [21], they require non-destructive-read-out memory. In binary SFQ, this type of memory uses NDROs (Table 1). Thus, the next challenge is to generate these SFQ pulse streams to encode the coefficients and weights in U-SFQ.

To generate the pulse streams, we adopt a Pulse Number Multiplier (PNM) that creates a high-speed pulse train from a low-frequency input [32, 46, 48]. Figure 9a shows the circuit and timing diagram for a typical PNM where a clock signal (CLK) is divided by a chain of TFFs. The output of each TFF is merged to generate the pulse stream. The number of pulses is programmable and set by the NDROs that each acts as an AND gate. Figure 9a shows the pulse stream (out) with the NDROs set to "1111", which yields 15 pulses. If instead the NDROs were set to "0100", the result is four pulses (S1).

The pulses generated by the PNM are not uniformly spaced to the detriment of computation accuracy. Recall that the pulses must appear at a regular frequency (Sections 3.2 and 4.1). To address this issue, we use a dual-port Toggle-flip-flop (TFF2), which works like a demultiplexer, splitting up a data stream into two signal lines [11]. One of the outputs of the TFF2 is used to divide the clock by half, while the other is used to form the pulse stream. The resulting pulse stream resembles a train of pulses with a uniform rate (Figure 9b).

Similar to an SFQ binary implementation, the NDROs in Figure 9b are the memory bank. To obtain pulse streams from this memory bank, we use the proposed PNM to generate the clock for the NDROs and mergers to form the pulse streams (Figure 9b). The mergers and clock distribution network cost a 10% area overhead compared to a binary implementation.

4.4 Shift Register

Another important piece to complete a U-SFQ accelerator is a shift register to enable an efficient memory implementation. In a typical binary implementation, the shift register is a bank of DFFs. Here we explore three possible implementations of a shift register for RL: (i) a combination of a binary shift register with binary-to-RL conversion; (ii) a DFF-based shift register for RL; and (iii) an integrator-based shift register for RL.

4.4.1 Binary to Race Logic Conversion. A straightforward implementation connects the binary bank of DFFs with binary-to-RL-converters (B2RC). B2RCs are programmable counters designed as an interleaved chain of TFFs and DFFs [22]. Figure 12 shows that this takes up to 3.2× more area than its binary counterpart due to the expensive converters.

4.4.2 DFF-Based RL Shift Register. To avoid expensive B2RCs, the shift register must have inputs and outputs encoded in RL. This can be achieved by connecting DFFs in series to create a delay chain controlled by a clock (Figure 10a). To delay a pulse by the total number of time-slots in an epoch, a DFF per time-slot is required. Since the number of DFFs grows exponentially with the number of bits, this solution is more expensive than using B2RCs (Figure 12).

4.4.3 Integrator-Based RL Shift Register. Therefore, we propose an integrator-based buffer for RL (Figure 10b). We use an inductor to integrate pulses from a clock source given the inductor's current-voltage relation $I_L = 1/L \int v_L dt$. The RL input pulse initiates the integration. One epoch later, the integration stops and generates the RL output.

In more detail, the arrival of a RL input (in) closes switch ①, starting the integration. The integration pauses when J1, which acts as a comparator, reaches its critical current I_c. The time it takes for J1 to kickback is equivalent to half an epoch. Then, the circuit starts to discharge the inductor by activating switch ② until it reaches a low baseline. At that time, J2 kicks-back generating the output pulse (out). The process of charging and discharging the inductor delays the input signal by a complete epoch. Figure 10c shows the SFQ circuit for the proposed buffer. The NDROs are switches ① and ②, while the DFFs take only the first pulse observed at the inductor ends L_a and L_b. Figure 11 shows the simulation waveforms for the buffer.

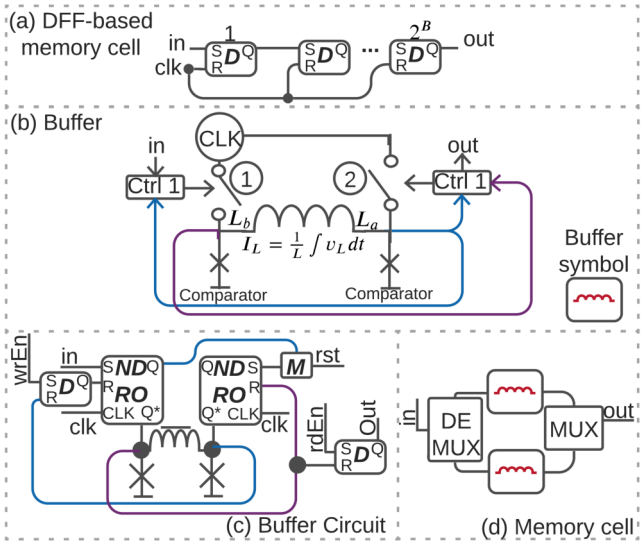

Figure 10: (a) Race logic shift register built using SFQ DFFs. (b) Conceptual integrator-based RL buffer using an inductance to accumulate SFQ pulses. (c) SFQ control logic for the integrator-based buffer. (d) Memory cell based on RL buffer.

A memory cell uses two buffers in parallel. While one buffer is delaying the input from the previous epoch, the other one receives the input from the current epoch. To interleave the two buffers, we use an RSFQ multiplexer and de-multiplexer [57] as shown in Figure 10d. The complete RL shift register is built by connecting memory cells in series.

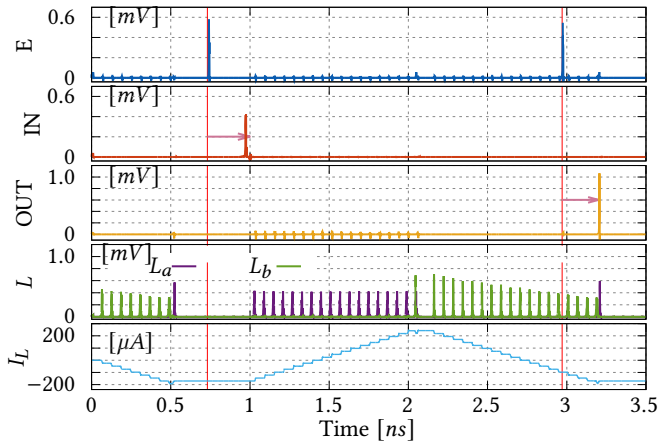

Figure 11: Simulated waveforms for the proposed buffer. E signals the beginning of a new epoch. The RL input pulse (IN) carries information in the time delay measured from the beginning of the epoch. The output pulse (OUT) appears with the same delay at the next epoch. Signals L_a and L_b are voltages at the inductor endpoints. I_L is the current in the inductor.

Figure 12 shows the area in JJs for the shift register built using our proposed buffer. In contrast to other implementations, the number of JJs for the buffer is constant while the inductance value increases with the number of bits. Our preliminary simulations show that the inductance increment is negligible compared with the JJ count of the other implementations. The inductor value L, the frequency of the clock, and the JJ's I_c depend on the bit resolution and epoch time.

All in all, the integrator-based shift register is smaller than B2RCs and the DFF-based RL options, yet it yields an area overhead compared to a binary shift register. For 8 bits, the area overhead is 2.5× but only 1.3× for 16 bits.

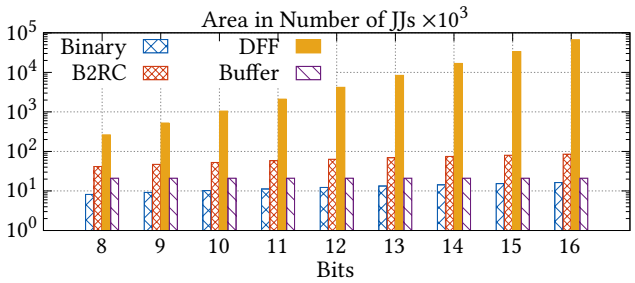

Figure 12: Number of JJs of different shift registers.

5 ARCHITECTURE EVALUATION

We evaluate our novel computing paradigm with three popular hardware accelerators: (i) A processing element (PE) suitable for CGRAs or spatial architectures. (ii) A DPU, popular in DSP and NN accelerators. (iii) An FIR accelerator where we also evaluate the impact of memory.

5.1 Methodology

We extract metrics of performance such as accuracy, latency, throughput, power consumption, area calculated as the number of JJs, and efficiency calculated as throughput per JJ (Figure 18). We run spice-level simulations using WRspice and the open-source *MIT-LL SFQ5ee 10 kA/cm2* process.

For DSP and error injection modeling we use Octave. We are interested in the range of bit resolutions and the number of taps where our U-SFQ architecture yields a performance advantage over its binary counterpart. The binary multipliers and adders are summarized in Table 2. The binary architecture uses a single multiplier and adder unit given the area limitations of RSFQ.

5.2 Processing Element

The Processing Element (PE) is the core of popular hardware accelerators such as CGRAs and SpA for CNNs [2]. Suitable kernels mainly perform repetitive operations, such as multiply accumulate (MAC) and arithmetic operations [2]. With the computing elements introduced above we can build a unipolar PE for CGRAs or SpAs as shown in Figure 13b. The proposed PE can perform the following mathematical operations:

- Unipolar multiplication with In1 a RL input and In2 a pulse stream (Section 4.1).
- Unipolar addition among In2 and In3 using the balancer described in Section 4.2 and setting In1 to 1.

- A multiply-accumulation operation using the integrator described in Section 4.4. The integrator performs two tasks. First, each pulse from the adder is integrated in the analog buffer (accumulation). Second, the accumulated result is returned in a RL format facilitating the interface among PEs.

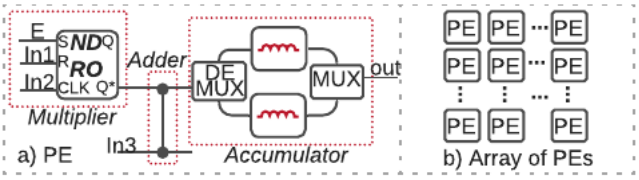

Figure 13: a) A simplified unipolar PE consisting of a multiplier, adder and integrator. b) Example of a typical array of PEs typically used in CGRAs and SpA.

The U-SFQ PE yields significant area savings when compared with its B-SFQ counterpart. The number of JJs for the U-SFQ PE is 126 and does not increase with the number of bits. On the other hand, the number of JJs for the binary PE increases linearly with the number of bits (Section 4). As an example, the U-SFQ yields 98%-99% savings in area when compared with an 8-bits B-SFQ PE that requires 9K-17k JJs.

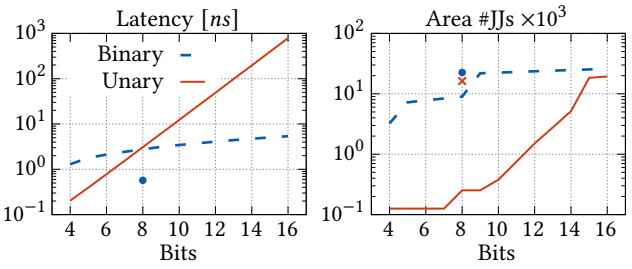

Figure 14: a) PE Latency. b) Area for the same throughput. Dashed-lines obtained from the metrics summarized in Table 2. •BP architecture [37, 38], × U-SFQ with a throughput equivalent to the BP architecture.

The minimum footprint of the U-SFQ PE comes at an increased latency. Figure 14a shows that in general the binary implementation yields better latency for an individual PE. However, the U-SFQ PE enables the integration of multiple PEs at a reduced area cost, thus increasing overall throughput. To evaluate this, we equalize throughput by increasing the number of U-SFQ PEs and then compare area (number of JJs) in Figure 14b. An array of U-SFQ PEs yields 93%-96% savings in area when compared with the Wave-Pipelined (WP) binary architecture for less than 12 bits. As the resolution increases the area savings reduce up to 30% for 16 bits. When compared with an 8-bit bit-parallel architecture [37, 38] there are area savings of 28%.

5.3 The Dot-Product Unit

The dot-product is the building block for several DSP algorithms [18] and for artificial neural networks [19]. Moreover, the dot-product is one of the most computationally expensive operations given the high number of multiplications required to obtain a result. These properties make the dot-product a good benchmark for our unary SFQ data representation. The dot product between vectors a and b of length L is defined as $y = a \cdot b = \sum_{i=0}^{L-1} a[i]b[i]$, where the result y is a scalar.

A straightforward mapping of the dot-product equation requires as many multipliers and adders as the input length L. Unfortunately, due to poor JJ device density, the number of binary multipliers and adders that can be practically deployed is restricted to 1-4 [21], especially for many bits.

By leveraging the small footprint of our proposed multiplier and adder (Sections 4.1 and 4.2), we can instantiate a larger number of them in parallel to implement a DPU. Figure 15 shows the proposed unary SFQ DPU. Our SFQ multiplier receives inputs $a_0, ...a_L$ in RL format while inputs $b_0, ..., b_L$ are pulse streams. The counting network receives parallel pulse streams and combines them such that $Y = \frac{a_0b_0+a_1b_1+...+a_Lb_L}{L}$

Figure 15: Proposed U-SFQ Dot Product Unit.

Figure 16 compares the area in number of JJs for the U-SFQ and SFQ binary implementations. The U-SFQ DPU area is independent of the number of bits and is proportional to the vector length L, while the binary DPU is proportional to both the number of bits and the vector length. The unary implementation yields area savings for L less than 64. For $L = 128$, both architectures become comparable and the area depends on the number of bits. For example, a unary DPU for a vector length of 128 yields area savings for a resolution of more than 12 bits. Finally, beyond 256 taps, the parallel multipliers and adders complexity increases beyond a single binary multiply accumulation unit.

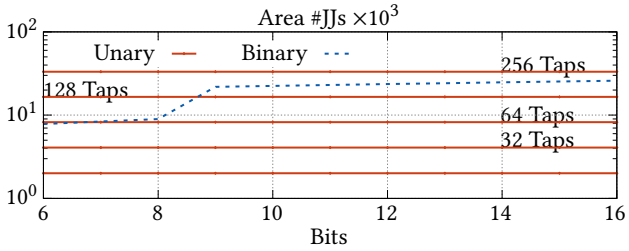

Figure 16: Area of the dot-product unit. For the unary DPU we vary the number of multipliers and counting network dimensions from 16 to 256. Dashed-lines obtained from the metrics summarized in Table 2.

5.4 The Unary SFQ Finite Impulse Response Filter (FIR)

Because of their stable and linear-phase response, FIR filters have widespread use in digital communication [53, 56], image processing, and signal preconditioning [24]. The output $y[n]$ of the FIR filter of order N is the dot product between the filter's impulse response $h(k)$ with the input vector $x(n-k)$ defined as $y[n] = \sum_{k=0}^{N-1} h(k)x(n-k)$.

A typical FIR architecture (Figure 17) has N taps. A tap consists of a multiplier, an adder, and a delay element z^{-1}. FIR filters are computationally intensive given the large number of multiplications. For example, Infra-Red (IR) sensors require 30 taps with 6-8 bits of resolution [3, 24, 42, 47], while Software-Defined-Radio (SDR) requires 200-900 taps and 7-14 bits of resolution [53, 56].

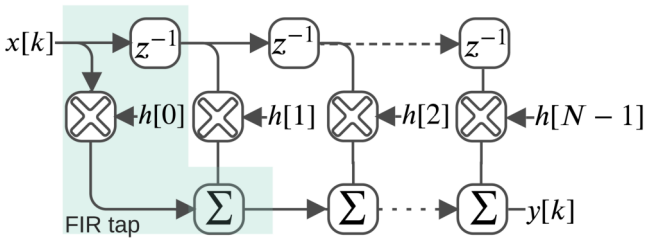

Figure 17: FIR block diagram.

To implement an FIR tap we can use the DPU unit for multiplication and addition and the RL shift register (Section 4.4) to implement the delay element z^{-1}. To store the coefficients $h[i]$, we use the memory bank introduced in Section 4.3. In practice, the format of external inputs and therefore any necessary conversions depends on the application. For instance, sensors could directly generate temporal inputs [16, 17]. For the FIR output, we could use an SFQ pulse counter to convert to binary representation. However, the circuit after our FIR may expect pulse streams (no need to convert) or RL. In the latter case, we can use the integrator of Section 4.4 to convert pulse streams to RL. In that case, the FIR latency is not affected and area increases by 50-200 JJs. As we discuss in this section, our proposed accelerator yields throughput and area advantages depending on the number of taps and bits resolution.

5.4.1 Accuracy. To evaluate the U-SFQ FIR accelerator response in the presence of errors, we use Octave to generate a golden reference that includes an input $x(t)$, the filter impulse response $h(t)$, and the filter output $y(t)$. The synthetic input $x(t)$ is a superposition of sinusoidal signals with frequencies at 1KHz, 7KHz, 8KHz, and 9KHz. We design a 16-taps FIR filter to recover the 1KHz sine wave from $x(t)$ and filter out the remaining higher frequency waves. Inputs are scaled to avoid overflow errors. The Signal-to-Noise-Ratio (SNR) of the sinusoidal obtained at the FIR filter output $y(t)$ is 25.7dBs. We build models for fixed-point binary and U-SFQ FIR filters. As expected, both the unary and binary architectures suffer an SNR degradation due to quantization noise. For instance, for 16 bits, the calculated SNR is $24dBs$ and for 6 bits is $15dBs$.

We then randomly inject errors in the computation results. For binary, an error typically results in bits flipped, causing the SNR to drop quickly as the number of errors increases (Figure 19). *i.e.*, three errors cause the SNR to drop $\sim 10dBs$ in average. The large SNR variance shows that the error can be catastrophic when the most significant bits flip.

Three possible errors can affect the U-SFQ FIR filter: **(i) Lost pulses in pulse streams:** A pulse can get lost due to non-ideal circuit behavior, *i.e.*, flux trapped in parasitic inductors, or delay variations causing pulse collisions in the adder (Section 4.2). **(ii) Lost pulses in RL:** Similarly, the RL pulse can be lost due to the aforementioned non-idealities of the circuit. **(iii) Delayed pulses in RL:** Either positive or negative delay variations cause the RL pulses to arrive outside the expected time-slot, modifying the value of the RL operand.

The effect of errors (i) and (iii) on the computation accuracy is similar. In contrast to a binary representation, each pulse in a pulse stream has the same weight $1/2^B$ where B is the bit resolution. Figure 19a shows that when the SNR for the binary implementation drops by 10dBs on average, the SNR for the U-SFQ filter only drops 4dB. For a binary implementation, the effect of the error depends on the bit weight as shown by the large distribution of SNR shown in Figure 19b. In the unary implementation, for $B = 16$, there are 2^B pulses in the stream, and each pulse weights 0.0000152. Thus, missing 30% of the pulses causes an SNR degradation of only 4dBs. A delay of ±30% in the RL input has a similar effect due to its interaction with the pulse stream. As discussed in Section 4.2, the adder returns the total number of pulses divided by two. When the number of pulses is odd, the result yields an error of ±0.5. Our model includes this effect in the accuracy evaluation.

Figure 19a also shows the effect of error (ii). A lost RL pulse has a larger effect on computation accuracy because all the information is concentrated in a single pulse. Given the low density of RL pulses in the FIR datapath, collisions are unlikely, in contrast to pulse streams. Special care should be taken during the layout of the RL lanes to avoid non-ideal effects such as unintentional flux trapped in parasitic inductors.

5.4.2 Performance. The latency for the U-SFQ FIR is set by the PNM described in Section 4.3. The period of the low-frequency clock used by the PNM is given by $T_{CLK} = t_{TFF2}B$, where $t_{TFF2} = 20ps$ is the delay of the TFF2 and B is the resolution in bits. Then, the total computation latency is $t = 2^B T_{CLK}$.

Figure 18a-b compares the latency and throughput[1] of the binary and unary FIR filters for 32 and 256 taps. The latency for the unary implementation is independent of the number of taps. Also, it yields latency and throughput advantages for less than 9 (12) bits with 32 (256) taps when compared with WP binary architectures. This range covers many on-sensor and edge computing applications, which is an area of increasing interest. Previous work in CNNs demonstrated that fixed-point arithmetic with as low as 4 bits yields acceptable accuracy [25]. The U-SFQ FIR yields better performance than the BP binary counterpart for 256-taps but not for 32-taps. This is because in the U-SFQ FIR the performance is set by the memory elements instead of the computing elements. The delay for the U-SFQ multiplier(adder) is $9ps(12ps)$ versus a PNM's delay of $20ps$.

[1]The operation in GOPs is the complete FIR computation.

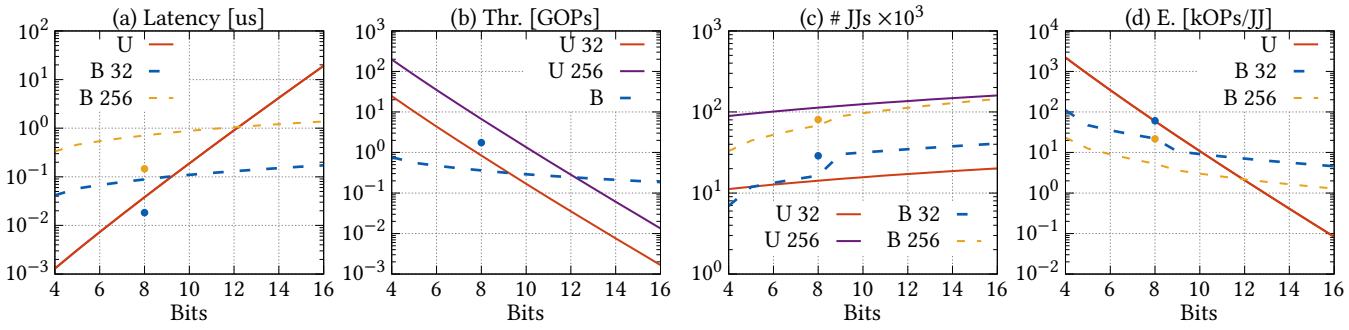

Figure 18: Unary vs binary FIR for 32 & 256 Taps. (a) Latency, (b) Throughput (c) Area in JJs, (d) Efficiency (throughput per JJ). Dashed-lines obtained from the metrics summarized in Table 2, • BP architecture [37, 38]

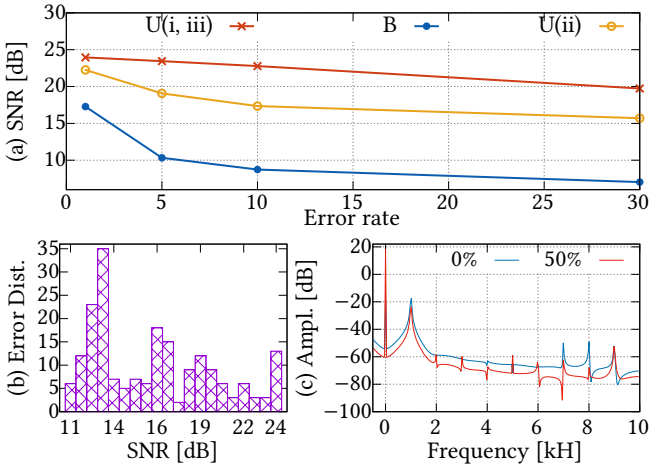

Figure 19: FIR accuracy evaluation. (a) SNR versus error rate. (b) Error distribution for an SFQ binary FIR with 1% of errors. (c) Effect of errors in the frequency response for U-SFQ FIR.

Figures 18a-b show that there is a design area where the unary architecture yields better performance than its binary WP counterpart. Figure 20a shows the area boundaries. The *x-axis* is the number of taps, the *y-axis* is the number of bits, and the *z-axis* shows in color the percentage of latency savings offered by the U-SFQ FIR accelerator. In white, we show where the binary architecture performs better. For example, an 8-bit, 32-taps U-SFQ FIR yields 56% of latency savings compared to its binary counterpart, yet increasing the resolution beyond 8 bits causes a latency penalty.

5.4.3 Area. Figure 18c compares the area of the WP binary and unary FIR filters for 32 and 256 Taps. Figure 20b shows in color where the unary architecture yields area savings and in white where the binary architecture performs better. For 32 taps, the minimum number of bits that yields area savings is 9, while for 256 taps, the unary implementation always requires more area due to the parallel adders and multipliers.

5.4.4 Efficiency Calculated as Throughput per JJ. We use throughput per JJ as a metric of efficiency. Figure 18d compares the efficiency of the binary and unary architectures for 32 and 256 taps. As before, Figure 20c shows where the U-SFQ FIR yields better efficiency than the WP binary counterpart. The U-SFQ FIR is more efficient for less than 12 bits. Moreover, the efficiency increases with the number of taps.

In Figure 20 we also highlight the design region of interest for two applications: SDR and IR sensors [3, 24, 42, 47, 53, 56]. Inside the SDR region, we place two commercial SDR cards as reference points. For an FIR that fits the RTL-2832U [44] card, the area of a unary implementation is 60% larger. However, it yields 80% better efficiency than its binary counterpart due to 90% lower latency. For IR sensors, the U-SFQ FIR accelerator yields 13%-78% savings in latency, 40% savings in area, and overall better efficiency of 62%-89% compared with the SFQ binary accelerator.

5.4.5 Power. We use WRspice simulations to extract power consumption for the building blocks introduced in Section 4. These simulations are set such that the activity factor is the average of the best and worst case scenarios. Since we combine RL and pulse streams, we obtain this average when the pulse streams are set to be half the maximum frequency, and the RL encoded inputs are set to be half the computing epoch.

Power consumption in RSFQ can be divided in two components. First, active power, which is in the order of tens of nWs per gate. Second, static power produced by the resistive bias current distribution network, which is typically in the order of μWs [21]. To evaluate the power efficiency of our circuits, we simulate the proposed unipolar PE (Section 5.2) using WRspice to obtain its power consumption. Without taking into account cooling costs, we find that the active power consumption is $0.8\mu W$ while the passive power is $262\mu W$. We also simulate the active power consumption for a 32 taps U-SFQ DPU obtaining $8.45\mu W$ (Table 3). Note that the active power consumption is three orders of magnitude smaller than CMOS ($\approx 1mW$).

The passive power consumption can be eliminated by using the Energy-efficient RSFQ (ERSFQ) and energy-efficient-synchronous-phase-compensation SFQ (eSFQ) [33, 54] logic families. These logic families replace the resistive biasing network with limiting JJs and series inductances [33], yielding the same performance (delay), at the cost of a slight (1.4×) increment in area and design cost [33]. Moreover, the cost of cryocooling can be bypassed for sensors, such as IR or x-ray detectors, that already operate at cryogenic temperatures [7].

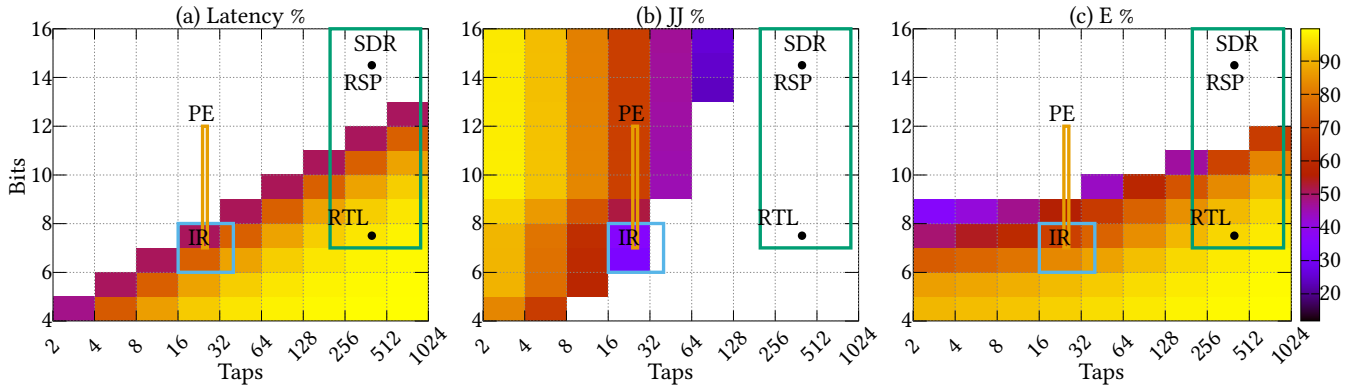

Figure 20: Unary vs WP binary FIR. Regions in colors show the unary gain %. Labelled areas in the plot mark the operation region for IR sensors and SDR.

Table 3: Power evaluation for a DPU with 32 multiplier/adders.

Component	Active $[mW]$	Passive $[mW]$
Multiplier	9×10^{-5}	0.05
Balancer	17×10^{-5}	0.1
DPU w/o cooling	84×10^{-4}	4.8

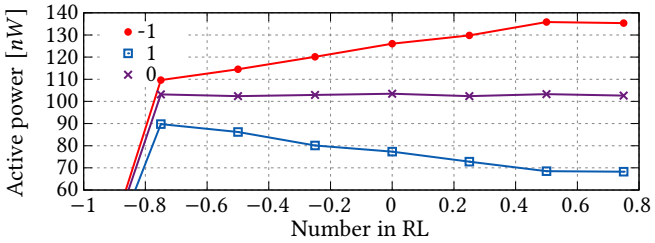

Figure 21: Active power consumption for the bipolar multiplier, using three different pulse streams frequencies representing the numbers -1, 1, and 0. We vary the RL input from -1 to 1.

To evaluate the effect of the activity factor (α) in the active power consumption, we run WRSpice simulations for the bipolar multiplier (Section 4.1) varying the RL input from -1 to 1. We also vary the pulse stream frequency from -1 to 1. Figure 21 shows the average active power consumption for minimum, maximum, and intermediate frequencies representing the numbers -1, 1 and 0 respectively. For -1 and 1 the number of pulses propagating from input to output increases and decreases respectively with the value of the RL input. This is reflected in the power consumption trends. For 0, the pulse stream frequency is set to half the maximum frequency, and the power consumption is constant because the same number of pulses propagate through the multiplier independently on the RL input. The multiplier's active power is bounded by a minimum power of $68nW$ and a maximum power of $135nW$.

6 RELATED WORK

Recent work in superconducting technology has demonstrated the potential for fast speed and low power operation despite cooling costs. For example, Nagaoka, *et al.* [37] demonstrated a superconducting pipelined 8-bit binary multiplier operating at 48GHz with a measured power consumption of 5.6mW and 17K JJs. Due to technology constraints, 17k JJs are already close to the maximum number of JJs that have been successfully manufactured. This area limitation restricts superconducting prototypes to small scales. For example Ishida *et al.* [21] demonstrated a superconducting chip with four 4-bit multiply-accumulation units and 8-bit 8-word shift registers. These prototypes are at odds with current CMOS circuits that need more than 200 processing elements with 6-16 bits of resolution to meet demanding requirements imposed by image processing, artificial neural networks, and SDR [21, 53, 56].

To address SFQ constraints, Cai, *et al.* [6], proposed a stochastic computing-based deep learning framework using adiabatic Quantum-Flux-Parametron technology. Also, Tzimpragos *et al.* [51] adapted RL to RSFQ. Our work introduces a novel U-SFQ architecture. The key in our U-SFQ is the data representation that combines the advantages of RL and pulse streams. We also introduce novel processing and memory elements required to build a complete architecture.

7 CONCLUSION

Superconducting SFQ technology is hindered by stringent area constraints. To address these challenges, we introduce the U-SFQ architecture that combines the advantages of pulse streams and RL data representations. We also introduce novel building blocks such as U-SFQ memory elements, multipliers and adders. Then, we propose and evaluate three hardware accelerators: a PE array, a DPU and an FIR filter. We find that our U-SFQ computing elements enable 28%-96% savings in area for the same throughput, yet, we expose the need for efficient memory elements to fully leverage the advantages of a U-SFQ architecture.

Data Availability Statement: We open-source a small DPU netlist, using a rudimentary testing environment [1].

ACKNOWLEDGMENTS

This work was supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE- AC02-05CH11231. The authors would also like to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] 2021. SuperC-FIR. https://zenodo.org/badge/DOI/10.5281/zenodo.5746945.svg. https://doi.org/10.5281/zenodo.5746945

[2] Omid Akbari, Mehdi Kamal, Ali Afzali-Kusha, Massoud Pedram, and Muhammad Shafique. 2020. X-CGRA: An Energy-Efficient Approximate Coarse-Grained Reconfigurable Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2020), 2558–2571. https://doi.org/10.1109/TCAD.2019.2937738

[3] Matthew Ash, Matthew Ritchie, and Kevin Chetty. 2018. On the application of digital moving target indication techniques to short-range FMCW radar data. *IEEE Sensors Journal* 18, 10 (2018), 4167–4175. https://doi.org/10.1109/JSEN.2018.2823588

[4] James Aspnes, Maurice Herlihy, and Nir Shavit. 1994. Counting Networks. *J. ACM* 41, 5 (sep 1994), 1020–1048. https://doi.org/10.1145/185675.185815

[5] Paul Bunyk, Konstantin Likharev, and Dmitry Zinoviev. 2001. RSFQ technology: Physics and devices. *International journal of high speed electronics and systems* 11, 01 (2001), 257–305. https://doi.org/10.1142/S012915640100085X

[6] Ruizhe Cai, Ao Ren, Olivia Chen, Ning Liu, Caiwen Ding, Xuehai Qian, Jie Han, Wenhui Luo, Nobuyuki Yoshikawa, and Yanzhi Wang. 2019. A Stochastic-Computing Based Deep Learning Framework Using Adiabatic Quantum-Flux-Parametron Superconducting Technology. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 567–578. https://doi.org/10.1145/3307650.3322270

[7] HS Cao and HJM Ter Brake. 2020. Progress in and Outlook for Cryogenic Microcooling. *Physical Review Applied* 14, 4 (2020), 044044. https://doi.org/10.1103/PhysRevApplied.14.044044

[8] Mikhail Dorojevets, Christopher Ayala, and Artur Kasperek. 2009. Development and evaluation of design techniques for high-performance wave-pipelined wide datapath RSFQ processors. In *Proc. ISEC*. 46.

[9] Mikhail Dorojevets, Christopher L Ayala, Nobuyuki Yoshikawa, and Akira Fujimaki. 2012. 16-bit wave-pipelined sparse-tree RSFQ adder. *IEEE transactions on applied superconductivity* 23, 3 (2012), 1700605–1700605. https://doi.org/10.1109/TASC.2012.2233846

[10] Mikhail Dorojevets, Artur K Kasperek, Nobuyuki Yoshikawa, and Akira Fujimaki. 2012. 20-GHz 8 x 8-bit parallel carry-save pipelined RSFQ multiplier. *IEEE transactions on applied superconductivity* 23, 3 (2012), 1300104–1300104. https://doi.org/10.1109/TASC.2012.2227648

[11] FG Theoretische Elektrotechnik. 2021. RFSQ cell library. https://www.tu-ilmenau.de/it-tet/forschung/supraleitende-hochgeschwindigkeits-elektronik/rsfq-cell/.

[12] Timur V. Filippov, Anubhav Sahu, Alex F. Kirichenko, Igor V. Vernik, Mikhail Dorojevets, Christopher L. Ayala, and Oleg A. Mukhanov. 2012. 20GHz Operation of an Asynchronous Wave-Pipelined RSFQ Arithmetic-Logic Unit. *Physics Procedia* 36 (2012), 59–65. https://doi.org/10.1016/j.phpro.2012.06.130 SUPERCONDUCTIVITY CENTENNIAL Conference 2011.

[13] Coenrad J. Fourie. 2018. Digital Superconducting Electronics Design Tools—Status and Roadmap. *IEEE Transactions on Applied Superconductivity* 28, 5 (2018), 1–12. https://doi.org/10.1109/TASC.2018.2797253

[14] B. R. Gaines. 1967. Stochastic Computing. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (Atlantic City, New Jersey) *(AFIPS '67 (Spring))*. Association for Computing Machinery, New York, NY, USA, 149–156. https://doi.org/10.1145/1465482.1465505

[15] Patricia Gonzalez-Guerrero and Mircea R Stan. 2019. Asynchronous Stochastic Computing. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 280–285. https://doi.org/10.1109/IEEECONF44664.2019.9049011

[16] Patricia Gonzalez-Guerrero, Tommy Tracy II, Xinfei Guo, Rahul Sreekumar, Marzieh Lenjani, Kevin Skadron, and Mircea R. Stan. 2020. Towards On-Node Machine Learning for Ultra-Low-Power Sensors Using Asynchronous Streams. *J. Emerg. Technol. Comput. Syst.* 16, 4, Article 44 (aug 2020), 20 pages. https://doi.org/10.1145/3404975

[17] Patricia Gonzalez-Guerrero, Stephen G Wilson, and Mircea R Stan. 2019. Error-latency Trade-off for Asynchronous Stochastic Computing with ΣΔ Streams for the IoT. In *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE, 97–102. https://doi.org/10.1109/SOCC46988.2019.1570548453

[18] Miao Hu and John Paul Strachan. 2016. Accelerating Discrete Fourier Transforms with dot-product engine. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 1–5. https://doi.org/10.1109/ICRC.2016.7738682

[19] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R Stanley Williams. 2016. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6. https://doi.org/10.1145/2897937.2898010

[20] IArpa. 2021. SuperTools program. https://www.iarpa.gov/research-programs/supertools.

[21] Koki Ishida, Ilkwon Byun, Ikki Nagaoka, Kosuke Fukumitsu, Masamitsu Tanaka, Satoshi Kawakami, Teruo Tanimoto, Takatsugu Ono, Jangwoo Kim, and Koji Inoue. 2020. SuperNPU: An extremely fast neural processing unit using superconducting logic devices. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 58–72. https://doi.org/10.1109/MICRO50266.2020.00018

[22] M Ito, N Nakajima, K Fujiwara, Nobuyuki Yoshikawa, A Fujimaki, H Terai, and S Yorozu. 2004. Design and implementation of SFQ programmable clock generators. *Physica C: Superconductivity* 412 (2004), 1550–1554. https://doi.org/10.1016/j.physc.2004.02.220

[23] Jin-Young Kim, Sehoon Kim, and Joonhee Kang. 2005. Construction of an RSFQ 4-bit ALU with half adder cells. *IEEE Transactions on Applied Superconductivity* 15, 2 (2005), 308–311. https://doi.org/10.1109/TASC.2005.849810

[24] Jongsun Park, Woopyo Jeong, H. Mahmoodi-Meimand, Yongtao Wang, H. Choo, and K. Roy. 2004. Computation sharing programmable FIR filter for low-power and high-performance applications. *IEEE Journal of Solid-State Circuits* 39, 2 (2004), 348–357. https://doi.org/10.1109/JSSC.2003.821785

[25] Karen Hao. 2021. Tiny four-bit computers are now all you need to train AI. https://www.technologyreview.com/2020/12/11/1014102/ai-trains-on-4-bit-computers/.

[26] Ryota Kashima, Ikki Nagaoka, Masamitsu Tanaka, Taro Yamashita, and Akira Fujimaki. 2021. 64-GHz Datapath Demonstration for Bit-Parallel SFQ Microprocessors Based on a Gate-Level-Pipeline Structure. *IEEE Transactions on Applied Superconductivity* 31, 5 (2021), 1–6. https://doi.org/10.1109/TASC.2021.3061353

[27] Naveen Kumar Katam, Jamil Kawa, and Massoud Pedram. 2019. Challenges and the status of superconducting single flux quantum technology. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1781–1787. https://doi.org/10.23919/DATE.2019.8747356

[28] Fei Ke, Olivia Chen, Yanzhi Wang, and Nobuyuki Yoshikawa. 2021. Demonstration of a 47.8 GHz High-Speed FFT Processor Using Single-Flux-Quantum Technology. *IEEE Transactions on Applied Superconductivity* 31, 5 (2021), 1–5. https://doi.org/10.1109/TASC.2021.3059984

[29] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. 2014. Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Minneapolis, Minnesota, USA) *(ISCA '14)*. IEEE Press, 517–528. https://doi.org/10.1145/2678373.2665747

[30] Vratislav Michal, Emanuele Baggetta, Mario Aurino, Sophie Bouat, and Jean-Claude Villegier. 2011. Superconducting RSFQ logic: Towards 100GHz digital electronics. In *Proceedings of 21st International Conference Radioelektronika 2011*. 1–8. https://doi.org/10.1109/RADIOELEK.2011.5936486

[31] George Michelogiannakis, Darren Lyles, Patricia Gonzalez-Guerrero, Meriam Bautista, Dilip Vasudevan, and Anastasiia Butko. 2021. SRNoC: A Statically-Scheduled Circuit-Switched Superconducting Race Logic NoC. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1046–1055. https://doi.org/10.1109/IPDPS49936.2021.00113

[32] Yoshinao Mizugaki, Jun Saito, Masataka Moriya, and Masaaki Maezawa. 2011. Design and operation of 64-fold variable single-flux-quantum pulse-number multiplier. *IEEE transactions on applied superconductivity* 21, 6 (2011), 3604–3607. https://doi.org/10.1109/TASC.2011.2166396

[33] Oleg A. Mukhanov. 2011. Energy-efficient single flux quantum technology. *IEEE Transactions on Applied Superconductivity* 21, 3 (2011), 760–769.

[34] Oleg A. Mukhanov, Stanislav V. Polonsky, and Vasili K. Semenov. 1991. New elements of the RSFQ logic family. *IEEE Transactions on Magnetics* 27, 2 (1991), 2435–2438. https://doi.org/10.1109/20.133710

[35] Alan F Murray. 1989. Pulse arithmetic in VLSI neural networks. *IEEE Micro* 9, 6 (1989), 64–74. https://doi.org/10.1109/40.42988

[36] Alan F Murray and Anthony VW Smith. 1988. Asynchronous VLSI neural networks using pulse-stream arithmetic. *IEEE Journal of Solid-State Circuits* 23, 3 (1988), 688–697. https://doi.org/10.1109/4.307

[37] Ikki Nagaoka, Masamitsu Tanaka, Koji Inoue, and Akira Fujimaki. 2019. A 48ghz 5.6 mw gate-level-pipelined multiplier using single-flux quantum logic. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 460–462. https://doi.org/10.1109/ISSCC.2019.8662351

[38] Ikki Nagaoka, Masamitsu Tanaka, Kyosuke Sano, Taro Yamashita, Akira Fujimaki, and Koji Inoue. 2019. Demonstration of an Energy-Efficient, Gate-Level-Pipelined 100 TOPS/W Arithmetic Logic Unit Based on Low-Voltage Rapid Single-Flux-Quantum Logic. In *2019 IEEE International Superconductive Electronics Conference (ISEC)*. IEEE, 1–3. https://doi.org/10.1109/ISEC46533.2019.8990905

[39] M Hassan Najafi, Shiva Jamali-Zavareh, David J Lilja, Marc D Riedel, Kia Bazargan, and Ramesh Harjani. 2017. Time-encoded values for highly efficient stochastic circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 5 (2017), 1644–1657. https://doi.org/10.1109/TVLSI.2016.2645902

[40] K Obata, M Tanaka, Y Tashiro, Y Kamiya, N Irie, K Takagi, N Takagi, A Fujimaki, N Yoshikawa, H Terai, et al. 2006. Single-flux-quantum integer multiplier with systolic array structure. *Physica C: Superconductivity and its applications* 445 (2006), 1014–1019. https://doi.org/10.1016/j.physc.2006.05.092

[41] M. Ozer, M. Eren Çelik, Y. Tukel, and A. Bozbey. 2014. Design of RSFQ wave pipelined Kogge–Stone Adder and developing custom compound gates. *Cryogenics* 63 (2014), 174–179. https://doi.org/10.1016/j.cryogenics.2014.05.007

[42] JENG-J Pan and Chein-I Chang. 1992. Destriping of Landsat MSS images by filtering techniques. *Photogrammetric engineering and remote sensing* 58 (1992), 1417–1417.

[43] Stanislav V. Polonsky, Vasili K. Semenov, and Alexander F. Kirichenko. 1994. Single flux, quantum B flip-flop and its possible applications. *IEEE transactions on applied superconductivity* 4, 1 (1994), 9–18. https://doi.org/10.1109/77.273059

[44] Realtek Semiconductor Corp. 2109. RTL2832u DVB-T COFDM DEMODULATOR + USB 2.0. https://www.realtek.com/en/products/communications-network-ics/item/rtl2832u.

[45] Peter Russer. 1971. General energy relations for Josephson junctions. *Proc. IEEE* 59, 2 (1971), 282–283. https://doi.org/10.1109/PROC.1971.8133

[46] Vasili K. Semenov. 1993. Digital to analog conversion based on processing of the SFQ pulses. *IEEE Transactions on Applied Superconductivity* 3, 1 (1993), 2637–2640. https://doi.org/10.1109/77.233969

[47] James J. Simpson, James R. Stitt, and David M. Leath. 1998. Improved Finite Impulse Response Filters for Enhanced Destriping of Geostationary Satellite Data. *Remote Sensing of Environment* 66, 3 (1998), 235 – 249. https://doi.org/10.1016/S0034-4257(98)00070-4

[48] Motohiro Suzuki, Masaaki Maezawa, Fuminori Hirayama, and Masayuki Ochiai. 2005. Design and operation of a pulse-number multiplier for a high-precision RSFQ D/A converter. *IEEE transactions on applied superconductivity* 15, 2 (2005), 336–339. https://doi.org/10.1109/TASC.2005.849827

[49] Swamit S. Tannu, Douglas M. Carmean, and Moinuddin K. Qureshi. 2017. Cryogenic-DRAM Based Memory System for Scalable Quantum Computers: A Feasibility Study. In *Proceedings of the International Symposium on Memory Systems* (Alexandria, Virginia) *(MEMSYS '17)*. Association for Computing Machinery, New York, NY, USA, 189–195. https://doi.org/10.1145/3132402.3132436

[50] Swamit S. Tannu, Poulami Das, Michael L. Lewis, Robert Krick, Douglas M. Carmean, and Moinuddin K. Qureshi. 2019. A Case for Superconducting Accelerators *(CF)*. 67–75. https://doi.org/10.1145/3310273.3321561

[51] Georgios Tzimpragos, Dilip Vasudevan, Nestan Tsiskaridze, George Michelogiannakis, Advait Madhavan, Jennifer Volk, John Shalf, and Timothy Sherwood. 2020. A Computational Temporal Logic for Superconducting Accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 435–448. https://doi.org/10.1145/3373376.3378517

[52] T. Van Duzer. 2005. Cryogenic Memories for RSFQ Ultra-High-Speed Processor. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC '05)*. IEEE Computer Society, USA, 66. https://doi.org/10.1109/SC.2005.21

[53] A Prasad Vinod and EM-K Lai. 2006. Low power and high-speed implementation of FIR filters for software defined radio receivers. *IEEE Transactions on Wireless Communications* 5, 7 (2006), 1669–1675. https://doi.org/10.1109/TWC.2006.1673078

[54] Mark H. Volkmann, Anubhav Sahu, Coenrad J. Fourie, and Oleg A. Mukhanov. 2013. Experimental Investigation of Energy-Efficient Digital Circuits Based on eSFQ Logic. *IEEE Transactions on Applied Superconductivity* 23, 3 (2013), 1301505–1301505. https://doi.org/10.1109/TASC.2013.2240755

[55] Di Wu, Jingjie Li, Ruokai Yin, Hsuan Hsiao, Younghyun Kim, and Joshua San Miguel. 2020. uGEMM: unary computing architecture for GEMM applications. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 377–390. https://doi.org/10.1109/ISCA45697.2020.00040

[56] Zhuan Ye, John Grosspietsch, and Gokhan Memik. 2007. An FPGA based all-digital transmitter with radio frequency output for software defined radio. In *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 1–6. https://doi.org/10.1109/DATE.2007.364561

[57] L Zheng, N Yoshikawa, J Deng, X Meng, S Whiteley, and T Van Duzer. 1999. RSFQ multiplexer and demultiplexer. *IEEE transactions on applied superconductivity* 9, 2 (1999), 3310–3313. https://doi.org/10.1109/77.783737

[58] Dmitry Zinoviev. 2021. RFSQ cell library. http://www.physics.sunysb.edu/Physics/RSFQ/Lib/contents.html.