A Computational Temporal Logic for Superconducting Accelerators

Georgios Tzimpragos
UC Santa Barbara
gtzimpragos@cs.ucsb.edu

Dilip Vasudevan
Lawrence Berkeley National Lab
dilipv@lbl.gov

Nestan Tsiskaridze
UC Santa Barbara
nestan@cs.ucsb.edu

George Michelogiannakis
Lawrence Berkeley National Lab
mihelog@lbl.gov

Advait Madhavan
National Institute of Standards and Technology & University of Maryland
advait.madhavan@nist.gov

Jennifer Volk
UC Santa Barbara
jevolk@ece.ucsb.edu

John Shalf
Lawrence Berkeley National Lab
jshalf@lbl.gov

Timothy Sherwood
UC Santa Barbara
sherwood@cs.ucsb.edu

Abstract

Superconducting logic offers the potential to perform computation at tremendous speeds and energy savings. However, a "semantic gap" lies between the level-driven logic that traditional hardware designs accept as a foundation and the pulse-driven logic that is naturally supported by the most compelling superconducting technologies. A pulse, unlike a level signal, will fire through a channel for only an instant. Arranging the network of superconducting components so that input pulses always arrive simultaneously to "logic gates" to maintain the illusion of Boolean-only evaluation is a significant engineering hurdle. In this paper, we explore computing in a new and more native tongue for superconducting logic: time of arrival. Building on recent work in delay-based computations we show that superconducting logic can naturally compute directly over temporal relationships between pulse arrivals, that the computational relationships between those pulse arrivals can be formalized through a functional extension to a temporal predicate logic used in the verification community, and that the resulting architectures can operate asynchronously and describe real and useful computations. We verify our hypothesis through a combination of detailed analog circuit models, a formal analysis of our abstractions, and an evaluation in the context of several superconducting accelerators.

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

Keywords superconducting logic; temporal logic; race logic.

ACM Reference Format:
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 (ASPLOS '20), March 16–20, 2020, Lausanne, Switzerland*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3373376.3378517

1 Introduction

Superconductivity is the phenomenon wherein the electrical resistance of a material disappears as it is cooled below a critical temperature. Computing with such superconducting materials offers the promise of orders of magnitude higher speed and better energy efficiency than transistor-based systems [11]. Unfortunately, while there have been tremendous advances in both the theory and practice of superconducting logic over the years, significant engineering challenges continue to limit the computational potential of this approach.

In contrast to semiconductor logic, where logic cells are combinational and their output is (to first order) a pure function of the levels of all the inputs present at any time, the majority of Single Flux Quantum (SFQ) logic [1] gates are sequential and operate on pulses rather than levels. Because pulses travel ballistically rather than diffusively through a channel, once they have transited there is no "record" of their value that can be used in downstream computations.

[1] SFQ technology and its variants have been dominant in superconductor digital technology for more than three decades with verified speeds of hundreds of GHz for simple digital circuits [4].

ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7102-5/20/03…$15.00
https://doi.org/10.1145/3373376.3378517

Implementing a chain of Boolean operations thus requires the very careful layout and *synchronization* of timing along each and every path with picosecond-level precision [27].

While some of the challenges in adopting such a novel technology are inherent to the nature of the exotic materials and environment, others appear to be due to a mismatch between our computational abstraction and what the devices actually provide. Because most superconducting logic designs rely on discrete voltage pulses driven by the transfer of magnetic flux quanta, supporting the combinational abstraction provided by traditional logic requires significant design effort and results in unavoidable overheads. If we instead think about these pulses as the natural representation of *data* in a superconducting system, the *natural language* for expressing computations over that data would be one that could precisely and efficiently describe the temporal relationships between these pulses. Here, we can draw upon two distinct lines of research, both currently disconnected from superconducting.

First, recent work has shown that delay-based encoding has both impressive computational expression and practical utility in implementing important classes of accelerators [19, 33] – not to mention the interesting connections to neurophysiology discovered by J. E. Smith [25]. The principles of the delay-coded logic described in that prior work apply directly to problems in superconducting. However, the fact that its primitive operators have been so far implemented only in CMOS under specific assumptions – e.g., edges are used to denote event occurrences – makes their realization in the much different RSFQ technology potentially challenging.

Second, we can leverage the long history of work in temporal logic used for expressing temporal relationships in reasoning and verification. While temporal logic systems (e.g. Linear Temporal Logic) deal with the relationship of events in time, they are fundamentally predicate logics that allow one to evaluate truth expressions (True / False) over some set of temporal relationships. We instead need a temporal logic with *computational* capabilities that takes events as inputs and creates new events as outputs based on the input relationships.

To explore these issues we present a new *computational temporal logic* (which in fact subsumes LTL) that gives clear, precise, and useful semantics to delay-based computations and relates them to existing temporal logics. This approach allows us to trade implementation complexity for delay, realize superconducting circuits that embody this new logic, and create useful new architectures based on these building blocks that encapsulate the potential of those circuits. To the best of our knowledge, this is the first time that temporal logic is used to specify computations. Overall, the main contributions of this paper are:

- We extend classical temporal predicate logic to a computational temporal logic to formally express delay-based computations. This extension provides the needed abstractions to capture the capabilities of our new operators and it sets the foundation for the construction, analysis, and evaluation of large-scale temporal systems.
- We design circuits that implement these primitive temporal operators in RSFQ and evaluate their functionality and performance with SPICE-level simulations.
- We describe a way of combining these temporal operators into larger self-timed superconducting accelerator architectures. Our data-driven self-timing approach enables the operation of our RSFQ designs without the need of clock trees even in the most general case.
- We validate our hypothesis through (a) a functional verification of three RSFQ accelerators at the SPICE level, (b) a performance comparison between our superconducting designs and their CMOS counterparts – showing more than an order of magnitude performance improvements – and (c) a timing analysis necessary to identify timing constraints that may affect the design flow of superconducting temporal accelerators.
- We open-source [2] our temporal primitives and accelerator designs for quick use and reference.

2 Background

2.1 Computing with Superconductors

Superconductivity was discovered in 1911 by K. Onnes, who observed that the resistance of solid mercury abruptly disappeared at the temperature of 4.2K [14]. Four decades later, D. A. Buck demonstrated the first practical application of this phenomenon – the cryotron [2] – and soon after, B. Josephson established the theory behind the Josephson effect [24], which led to the fabrication of the first Josephson junction (JJ) in the subsequent years.

A JJ is made by sandwiching a thin layer of non superconducting material – an electronic barrier – between two layers of superconducting material. JJs are capable of ultrafast (as low as 1ps), low-energy (to the order of 10^{-19}J) switching by exploiting the Josephson effect: electron pairs tunnel through the barrier without any resistance up to a critical current. At the critical threshold, a JJ switches from its superconducting state to a resistive one and exhibits an electronic "kickback" in the form of magnetic quantum flux transfer – observable as a voltage pulse on the output. To enable stateful circuit operation the unit of flux can be temporarily stored in a composite device known as the superconducting quantum interference device (SQUID), which is built as a superconducting loop interrupted by two serial JJs and is common to many superconducting circuits.

[2] https://github.com/UCSBarchlab/Superconducting-Temporal-Logic

Over the years, several ambitious designs of superconducting ALUs [29, 30] and microprocessors [1, 7, 34] have been presented in an effort to capitalize on the promise of superconductors [12]. The majority of these implementations are primarily based on simplified architectures, bit-serial processing, and on-chip memories realized with shift-registers. Bit-serial processing has been selected over bit-parallel approaches due to its lower hardware cost and complexity. However, this design choice compromises the advantage speed of SFQ technology [4] as the number of execution cycles per instruction increases with the number of bit slices. Moreover, the use of shift register-based memories – given the lack of dense, fast, and robust cryogenic memory blocks – seems to be the only reasonable choice at the moment; still not a viable solution though for large-scale designs [3].

More recently, interest has increased in the development of superconducting computing accelerators. As stated by S. S. Tanu, et al. [31], due to the lack of sophisticated design tools and the limited device density and memory capacity in superconducting technology, applications with tiny working set sizes and high computational intensity are ideally suited for JJ-based accelerators. As a proof-of-concept, the authors developed an RQL-based accelerator for SHA-256 engines, achieving 46× better energy-efficiency than CMOS. To improve the critical path and the overall energy efficiency of their implementation, the optimization focus was on the two most critical components of the SHA engine: adders and registers. However, no answers were given to questions of more general interest.

Another promising effort is the stochastic computing-based deep learning acceleration framework presented by R. Cai, et al. [3]. The authors of this work took advantage of stochastic computing's time-independent bit sequence value representation and the small hardware footprint of its operators to redesign the basic neural network components in AQFP and were able to achieve orders of magnitude energy improvements compared to CMOS. However, the known drawbacks of stochastic computing [21] (e.g., the calculation accuracy, expressiveness, and performance of stochastic computing circuits depend on the length and correlation of the used bit-streams) raise a number of questions regarding the suitability and efficiency of this method for more general tasks or for precise computing applications.

While these implementations succeed at demonstrating the potential of superconducting computing, the question of "what a more general superconducting design methodology would look like?" is still pending. To get a better understanding of the reasons that make superconducting computing so challenging a good idea may be to take a step back and look closer at the fundamentals of this technology as well as its main differences from CMOS [16, 17].

In contrast with CMOS, where an "1" is represented by a steady voltage level in hundreds of millivolts, in SFQ, picosecond-duration, millivolt-amplitude *pulses* are used. Moreover, SFQ comes with a different set of active (JJs) and passive (inductors) components and interconnection structures (Josephson Transmission Lines and Passive Transmission Lines) than CMOS. Clock distribution and synchronization are also major concerns [10] as each Boolean SFQ logic gate has to be driven by a synchronous clock and all input pulses need to be aligned.

Given the difficulties that existing approaches face and the unique characteristics of superconducting technology, the most promising way forward is to come up with innovative computing paradigms and circuit architectures that (a) use much fewer JJs than transistors for the same information processing, (b) have low memory requirements, (c) allow for easier clocking, and (d) can cover a wide range of applications [32]. Race logic provides exactly this opportunity.

2.2 Race Logic

The core idea behind race logic [19, 33] is to encode information in the timing of events rather than the amplitude of voltage levels. Events are represented by low to high edges and computation emerges through the purposeful interaction of these edges and their relative delays. The time it takes for an event to appear on a wire is what encodes the value. Thus, only a single wire is required per variable.

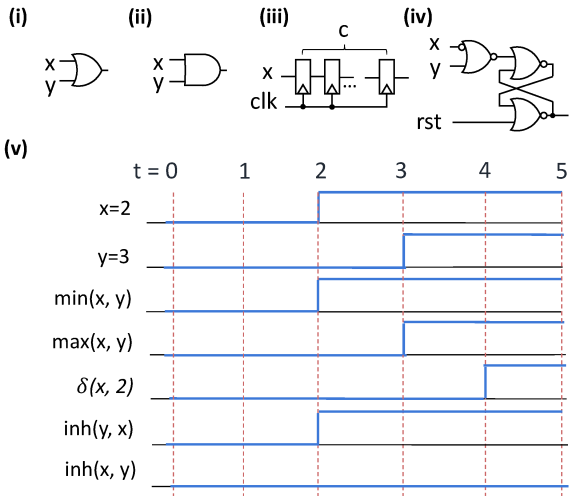


Figure 1. Panels (i), (ii), (iii), and (iv) show the implementation of Min, Max, Add-Constant, and Inhibit functions in race logic with off-the-shelf CMOS components. Panel (v) provides an example waveform for $x = 2$ and $y = 3$.

The operators forming the foundation of race logic are Min (FirstArrival), Max (LastArrival), Add-Constant

[3] State-of-the-art RSFQ processor implementations assume 256-bit on-chip memories [1]. However, even relatively small changes in their size can lead to a significant increase in access delay and JJ count [28] making them impractical for real-world applications.

(Delay)[4], and Inhibit[5]. Figure 1 shows the implementation of these four primitives with off-the-shelf CMOS components. Prior to the next computation, race logic-driven circuitry must be *reset*.

Regarding its applicability, race logic yields a complete implementation of space-time algebra [25, 26], which provides a mathematical underpinning for temporal processing. Any function that satisfies the properties of *invariance* and *causality* complies with space-time algebra, and thus it is implementable in race logic. In the past, A. Madhavan, et al. [19] used race logic to implement Needleman and Wunsch's popular DNA sequence algorithm. M. H. Najafi, et al. [23] demonstrated a low-cost bitonic sorting network circuit using temporal processing. G. Tzimpragos, et al. [33] applied race logic to accelerate ensembles of decision trees, while J.E. Smith [25] explored the relationship between temporal codes and spiking neural networks.

Besides its promises though, the implementation of this new paradigm, where the order of events occurrence defines computation, is currently tied to specific assumptions and the properties of the underlying CMOS technology, which in some cases may restrict innovation. For example, as discussed above, when edges are used for event representation, Min and Max functions can be realized with plain OR and AND gates. What happens though when edges are replaced by pulses, as in the superconducting case? To answer this question and establish a theoretical foundation that will allow us to better understand how processing in the temporal domain can unlock the true potential of emerging technologies we proceed with this logic's formalization.

3 Formalization

Computing based on temporal relationships departs from the traditional binary encoding and Boolean logic, and provides a promising pathway for unlocking the true potential of emerging technologies. However, to make this computing paradigm a viable solution the first question that we should answer is "what abstractions do we need to establish in order to capture its capabilities, verify the correctness of temporal implementations independently of underlying assumptions and technology properties, and build more complex temporal circuits in a systematic way?".

To solve this problem and set the foundation for the design and evaluation of large-scale temporal systems, in this section, we provide formal definitions of its primitive operators and constraints through an extended temporal logic capable of concisely expressing delay-based computations.

3.1 Computational Temporal Logic

Space-time algebra [25, 26] defines the primitive operators of *generalized race logic* over the set of *Natural* numbers, and thus provides a high-level abstraction to the event-based computation happening at the circuit-level. This abstraction may in some cases be useful for functional interpretation or synthesis; however, it cannot capture lower-level details that may be critical for the hardware implementation and reasoning of such systems. Our aim is to build a formalization that covers this gap and safely decouples functional from implementation specifications.

Temporal logic is a tool commonly used for representing and reasoning about propositions qualified in terms of time; e.g., an event in a system S has happened or will happen some time in the past or future. A system S transitions through a sequence of states in time, where each state S_t is associated with a time step t belonging to a discrete time domain. Properties are then expressed as formulas and are evaluated along such sequences of states. Formulas are constructed recursively from propositional atoms by applying usual propositional connectives $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ and the additional temporal logic operators discussed below.

In the well-established setting of Linear Temporal Logic (LTL), the future-time temporal operators are used: ◊ *some time in the future*, □ *always in the future*, ○ *next time* (*tomorrow*), **U** *until*, and **R** *release*. Past LTL (PLTL) extends LTL with past-time operators, which are the temporal duals of the future-time operators, and allows one to express statements on the past time instances, such as: ♦ *sometime in the past*, ■ *always in the past* (*historically*), ● *previous time* (*yesterday*), **S** *since*, and **T** *trigger* [5, 9, 15]. Even though the past-time operators do not add expressive power in the sense that any LTL formula with past operators can be rewritten by only using the future-time temporal operators, the past-time operators are particularly convenient in practice; they allow us to keep specifications more intuitive and easy to comprehend, and they can provide significantly more compact representations than their future-time counterparts.

Each operator operates on a sequence of states, which defines a discrete interval of timesteps – the *scope* of the operator. We categorize these temporal operators based on their *scope* at time step t as follows:

- *remote past* operators ♦, ■, **S**, **T** – their scope is $[0, t]$;
- *immediate past* operator ● – its scope is $\{t - 1\}$, or the empty interval if $t = 0$;
- *present* operator (all propositional connectives) – its scope is $\{t\}$;
- *immediate future* operator ○ – its scope is $\{t + 1\}$;
- *remote future* operators ◊, □, **U**, **R** – their scope is $[t, \infty)$.

[4] We assume that smaller delays in rise time encode smaller magnitudes, while larger magnitudes are encoded as longer delays. In the case where shorter delays represent larger magnitudes, FirstArrival will stand for Max, LastArrival will serve as Min, and Delay as Constant Subtraction.

[5] The Inhibit function has two inputs: an inhibiting signal and a data signal. If the data signal arrives first, it is allowed to pass through the gate unchanged. Otherwise, the output is prevented from ever going high, which corresponds to ∞ in the race logic world.

The scope of an arbitrary formula ϕ is defined recursively based on the scopes of the operators in ϕ and the given time step t.

In LTL, the notation $\langle S, t\rangle$ is used to signify a system S at time step t. We say that an *event* ϕ *occurs* at time step t in the system S, if ϕ holds at time step t in S, denoted by $\langle S, t\rangle \models \phi$. In this paper, we primarily rely on the formal semantics of the $\blacklozenge$ operator (*sometime in the past*):

$$\langle S, t\rangle \quad \models \quad \blacklozenge\phi \quad \text{iff} \quad \exists k.\, (0 \le k \le t \wedge \langle S, k\rangle \models \phi)$$

This definition reads as: the temporal formula $\blacklozenge\phi$ holds at time step t in the system S if and only if there exists a time step k prior or equal to t when the formula ϕ holds. However, this operator is incapable of encapsulating *when* ϕ held in the past, which is essential for our case. To address this issue, we introduce the *earliest-occurrence* function described below.

Let ∞ be a special symbol that represents an *unreachable* time step; in other words, ∞ indicates the lack of an event occurrence in a period of interest. The earliest-occurrence function $\mathcal{E}_{\langle S,t\rangle}(\phi)$ receives as input a formula ϕ and returns the *earliest* time step $t_{min} \in [\![\phi]\!]_{\langle S,t\rangle}$, where $[\![\phi]\!]_{\langle S,t\rangle}$ is the scope of ϕ at time step t in the system S, such that $\langle S, t_{min}\rangle \models \phi$. If ϕ does not hold at any time step within $[\![\phi]\!]_{\langle S,t\rangle}$, then the earliest-occurrence function returns ∞. The formal definition of this function follows:

$$\mathcal{E}_{\langle S,t\rangle}(\phi) = \begin{cases} t_{min} & (t_{min} \in [\![\phi]\!]_{\langle S,t\rangle}) \wedge (\langle S, t_{min}\rangle \models \phi) \wedge \\ & \wedge\, (\forall j. 0 \le j < t_{min} : \langle S, t_{min}\rangle \not\models \phi) \\ \infty, & \text{otherwise.} \end{cases}$$

The proposed function is paired with the existential primitives of the classical temporal logic, extends the notions of "some time in the past" and "some time in the future" with the notion of "when" an event occurred, and it is fundamental for the connection of our event-based formalization, which we present next, with the existing space-time theory.

3.2 Race Logic Semantics

According to space-time algebra [25, 26], FirstArrival ($\mathbf{FA}$), Inhibit ($\mathbf{I_S}$), and Delay ($\mathbf{D}$) operators are functionally complete for the set of space-time functions. In prior work, the functionality of these operators at the event-level has been primarily described through their realization with off-the-shelf CMOS components under the assumption of an edge-based delay encoding. In this work, we decouple for the first time their specification from their implementation and provide a formal definition, presented in Table 1, using the above-described *computational* temporal logic. Moreover, besides these three basic operators, we provide definitions for LastArrival ($\mathbf{LA}$) and Coincidence ($\mathbf{C}$) operators, which have been widely used in a number of accelerators.

Table 1. PLTL-based semantics of the operators FirstArrival ($\mathbf{FA}$), StrictInhibit ($\mathbf{I_S}$), Delay ($\mathbf{D}$), LastArrival ($\mathbf{LA}$), and Coincidence ($\mathbf{C}$).

$$\begin{array}{lcll}
\langle S, t\rangle & \models & \mathbf{FA}\phi\psi & \text{iff } \langle S, t\rangle \models \blacklozenge\phi \vee \blacklozenge\psi \\
\langle S, t\rangle & \models & \psi\mathbf{I_S}\phi & \text{iff } \exists k.\, (0 \le k \le t \wedge \langle S, k\rangle \models \phi \wedge \neg\blacklozenge\psi); \\
\langle S, t\rangle & \models & \mathbf{D}c\phi & \text{iff } \exists k.\, (0 \le k + c \le t \wedge \langle S, k + c\rangle \models \blacklozenge\phi); \\
\langle S, t\rangle & \models & \mathbf{LA}\phi\psi & \text{iff } \langle S, t\rangle \models \blacklozenge\phi \wedge \blacklozenge\psi; \\
\langle S, t\rangle & \models & \mathbf{C}\phi\psi & \text{iff } \exists k.\, (0 \le k \le t \wedge \langle S, k\rangle \models \phi \wedge \psi) \wedge \\
 & & & \quad \wedge \forall j. (0 \le j < k \wedge \langle S, j\rangle \not\models \blacklozenge\phi \vee \blacklozenge\psi);
\end{array}$$

Informally, Table 1 reads as follows:

- $\mathbf{FA}$: the formula $\mathbf{FA}\phi\psi$ holds at time step t in system S if and only if either ϕ or ψ hold at time step t or prior.
- $\mathbf{I_S}$: the formula $\psi\mathbf{I_S}\phi$ holds at time step t in system S if and only if there exists a time step k prior or equal to t when ϕ holds and ψ does not hold at this and any prior time steps.
- $\mathbf{D}$: the formula $\mathbf{D}c\phi$ holds at time step t in system S if and only if there exists a time step $k + c$ prior or equal to t when ϕ holds at this ($c = 0$) or any prior time steps ($c \neq 0$).
- $\mathbf{LA}$: the formula $\mathbf{LA}\phi\psi$ holds at time step t in system S if and only if both ϕ and ψ hold independently at time step t or prior.
- $\mathbf{C}$: the formula $\mathbf{C}\phi\psi$ holds at time step t in system S if and only if there exists a time step k prior or equal to t when both ϕ and ψ hold simultaneously and there are no prior time steps where either ϕ or ψ hold.

These definitions provide a PLTL-based specification of the basic race logic operators over temporal events; however, they will always return a proposition: *True* or *False*. To extract the step at which these functions evaluate to *True* for the first time in their scope the above-introduced earliest-occurrence function $\mathcal{E}_{\langle S,t\rangle}(\phi)$ has to be used. For example, $\mathcal{E}_{\langle S,t\rangle}(\mathbf{FA}\phi\psi)$ will return the first time step that either ϕ or ψ hold.

In a nutshell, the presented formalism, along with the proposed extension to the classical temporal logic: (a) guarantees that the specification of our operators is independent of any underlying assumptions; e.g., pulse- vs edge-based encoding, (b) bridges the gap between the high-level definitions provided by space-time algebra and the event-based computing happening at the implementation level, and (c) opens up the door to the use of model checking tools for the formal analysis, validation, and optimization of more complex temporal circuit designs.

4 Superconducting Temporal Architecture

The mathematical formalism raised in Section 3 lays the foundation for building and verifying the desired temporal operators. In this section, we first describe their implementation in RSFQ, then we present the corresponding circuit simulation results, and finally we propose a self-clocked RSFQ architecture that alleviates the clock distribution and skew problems met in traditional digital designs ported from CMOS to the RSFQ world.

4.1 Temporal Primitives in RSFQ

The way in which events are encoded plays a critical role in selecting the hardware that most efficiently implements logic operators. For example, given the conventional rising edge-based realization of events, FirstArrival and LastArrival functions can be implemented with a single OR and AND gate, respectively. As shown in Figure 1, an OR gate fires when its first high input arrives, while an AND gate fires only when all its inputs are "1". An important property of edge-based event encoding is that it automatically keeps track of the input state at all times – a signal that has made a transition from a "low" to "high" state will not make a transition back to a "low" state in the same computation. This feature *breaks down* when dealing with pulses. Pulses naturally return back to their "low" state, preventing downstream nodes from implicitly knowing the state of its predecessors.

To address this issue we propose embedding the state into each gate, instead of relying on the input to "hold" state for us. Interestingly, the majority of RSFQ elementary cells have both logic and storage abilities [18], and thus they provide several unique design opportunities. In Figure 3, we present the schematics of our circuit designs, along with Mealy machines describing their operation, and WRSPICE [13] simulations showing their functionality. A detailed description of their implementation also follows.

According to its formal definition, the FirstArrival gate $\mathbf{FA}$ emits an output pulse when its first input arrives. For its implementation in RSFQ, a Merge element along with a D flip-flop are used – Figure 2 (i) and Figure 3 (i). A Merge element can be thought of as a non-latching OR gate that produces an output SFQ for each incoming pulse from any of its input ports. However, in race logic, at most one event is allowed to occur per "wire" across the entire computation. To ensure that all but the first arriving pulses will be filtered out a D flip-flop has to be used. A D flip-flop is built around a direct current (DC) SQUID and has two stable states: *Init* and *Loaded*, which correspond to the lack or presence of a flux quantum, respectively. When a data signal arrives at its input port the latch switches to/remains in the *Loaded* state and returns to the *Init* state only when a clock signal is received. When transitioning from *Loaded* to *Init* the flux quantum stored in the quantizing SQUID loop is released; thus, an output pulse is emitted and the quantizing loop gets cleared. While in state *Init*, a clock pulse will not cause any state change or output activity. So, to achieve the desired functionality a *reset* signal *rst* is connected to the D flip-flop's data "in" port, while the output of the Merge element plays the role of the clock signal.

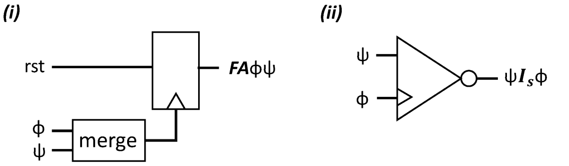


Figure 2. Panel (i): FirstArrival gate is built out of a Merge element and a D flip-flop; a *reset* signal *rst* is connected to the D flip-flop's *data* input, while the output of the Merger serves as its *clock* signal. Panel (ii): to implement Inhibit a latching Inverter is used; the data signal ϕ serves as the Inverter's *clock* signal and the inhibiting signal ψ as its *data* signal.

The Inhibit operator $\mathbf{I_s}$ receives two input signals: one for the inhibiting signal ψ and one for the data signal ϕ. As described in Section 2.2, an output pulse is emitted only if ϕ arrives before ψ. To implement Inhibit in RSFQ we use a single Inverter – Figure 2 (ii) and Figure 3 (ii). According to the Inverter's specification, if a data pulse arrives, the next clock pulse reads out "0"; otherwise, it reads out "1". Thus, if we route signal ϕ to the inverter's clock port and ψ to its data port, this component will act exactly as an Inhibit operator in our logic.

In traditional SFQ circuits, Josephson Transmission Lines (JTLs) are commonly used for the interconnection of logic cells over short distances. More specifically, a JTL is a serial array of superconducting SQUIDs and operates in the following way. Because magnetic flux cannot be absorbed or dissipated by a superconducting circuit, an incident flux quantum is only allowed to pass along the JTL, and does so by switching each JJ in turn. In our case, these interconnection structures are not used just for pulse transmission purposes but also realize our Delay operator $\mathbf{D}$ – Figure 3 (iii). As described in Section 2.2, delaying a race logic event by a fixed amount of time corresponds to Constant Addition.

For the implementation of the LastArrival gate, a C-element is used – Figure 3 (iv). A C-element has two input ports and consists of two SQUIDs. In the circuit's initial state, no persistent superconducting current is present in the quantizing loops. When an input arrives, the corresponding junction gets triggered but the generated pulse is not sufficient to trigger an output pulse. When the second input pulse arrives, the total current exceeds the threshold of criticality, an output pulse gets emitted, and the element returns to its initial state. The order of input pulse arrivals does not matter.

Finally, a Coincidence gate is supposed to fire only if its inputs arrive "simultaneously". In edge-based implementations, a Coincidence gate is composed of $\mathbf{FA}$, $\mathbf{LA}$, $\mathbf{I_s}$, and D gates. In the pulse-based superconducting logic though, a single RSFQ AND gate is all needed to implement Coincidence – Figure 3 (v). As known, an RSFQ AND gate produces an

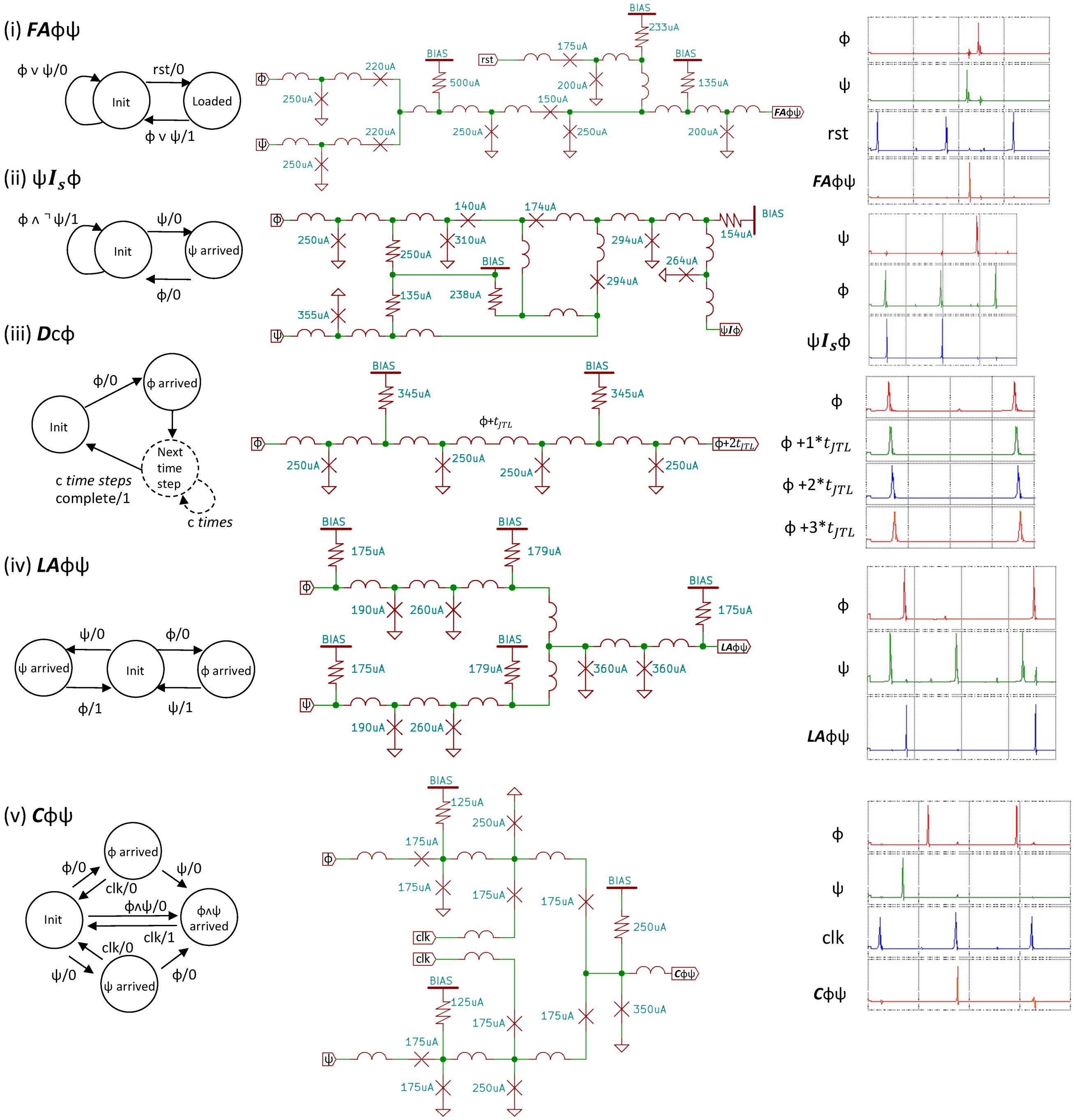


Figure 3. Panel (i): FirstArrival FAab. Panel (ii): strict Inhibit $a\mathbf{I}_S b$. Panel (iii): Delay Dca, where t_{JTL} is the delay that each JTL causes. Panel (iv): LastArrival LAab. Panel (v): Coincidence Cab.

output pulse only if both its input pulses arrive within the same cycle; thus, it performs in an easy way the desired functionality.

Area and latency results for each of these operators are provided in Table 2. The shown estimates are based on our WRSPICE [13] simulations using the MIT-LL SFQ5ee 10 kA/cm^2 process.

Table 2. Area and latency results for our temporal operators implemented in the MIT-LL SFQ5ee 10 kA/cm^2 process and simulated with WRSPICE [13].

Function	Area (#JJs)	Latency (ps)
FA	10	13
$\mathbf{I_s}$	8	11
D	2/JTL	5/JTL
LA	6	8
C	11	9

4.2 Self-clocked Temporal RSFQ Circuits

Clocking and synchronization are two of the most critical concerns and limitations in the design process of an RSFQ design. The majority of RSFQ Boolean gates are sequential in nature. Hence, each gate in a Boolean RSFQ circuit needs to be synchronized with all other gates and the clock network.

The complexity and overhead introduced by the clock network are far from negligible, primarily because an additional Splitter is required for each latched gate for clock fan-out[6]. These additional Splitters affect a design both in terms of area (3 JJs per element) and speed (each Splitter introduces a delay on the order of a single JTL), while they also contribute to a higher static and dynamic current [10, 27].

Moreover, device variations can promote disproportionate clock timing skews, which can critically affect the functionality of a Boolean RSFQ design; all pulses between a gate and each of its fan-in gates must arrive in the same clock cycle (as defined by the clock network). To mitigate these issues, advanced path-balancing techniques and customized RSFQ logic synthesis tools are needed.

In our superconducting temporal logic, many of these concerns are naturally alleviated as FirstArrival, Inhibit, and Delay, which form the minimal functionally complete operator set, are asynchronous – the "clock" signal of the latching building blocks used for their realization has been repurposed, as described in Section 4.1. However, in some cases, such as Coincidence, the use of a synchronous gate/block makes sense. To avoid costly clock trees and the clock skew problems that come with them, we propose a *data-driven self-timing* scheme.

In a data-driven self-timed (DDST) system, timing information is carried by data. Z. J. Deng, et al. [6] explored for the first time such an idea, targeting binary RSFQ circuits, more than two decades ago. In their solution, data are carried by complementary signals, generated by using complementary D flip-flops; two parallel lines are required for each bit. The clock signal is generated by a logical OR function between these lines. Because each functional block is now locally clocked, there is no need for a global clock network. Therefore, the system becomes more robust to process variations and has better control over clock timing [7]. Besides its advantages though, this idea never really took off due its high cost; the method introduces a significant overhead for routing as well as additional circuitry for generating complementary signals for each logic gate.

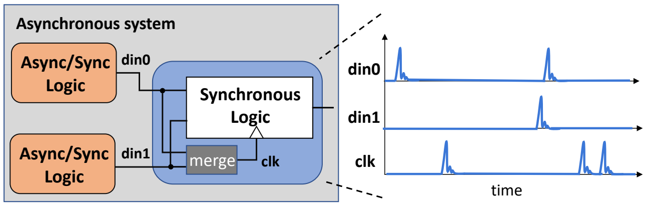


Figure 4. Proposed data-driven self-timing (DDST) scheme. The clock signal can be locally generated from input data at each gate. If no input pulse arrives, it is safe to assume the operator idle, and thus no clock pulse is required.

Our DDST method – shown in Figure 4 – targets temporal rather than binary systems and is able to provide similar benefits at a much lower "price". In contrast to Boolean logic, where for example a Not gate has to be clocked even in the absence of an incoming pulse, when processing in the temporal domain, an operator can be safely considered idle for the time steps that no input pulse arrives. Thus, complementary data are no longer required. This characteristic of temporal codes significantly simplifies the implementation of our DDST approach, reduces its area overhead, while still allowing one to have the desired fine-grained timing control.

Resetting: Given the absence of an independent clock and the stateful nature of RSFQ operators, resetting must also be rethought. For example, an RSFQ Inverter, which implements Inhibit, will not return to its initial state until a pulse arrives to its clock port, while a C-element, used as a LastArrival gate, will not reset until both its inputs arrive.

One possible solution is to add an additional reset signal to $\mathbf{I_s}$ and **LA** gates, merged with their input data signals; as can be seen in Figure 3, the rest of our temporal operators return to their initial state without the need for external signals. This additional signal allows the immediate reset of such a gate but it comes with additional circuitry too. For example, the Merger that has to be used in the case of Inhibit will cost us at least 5 JJs, while the overhead in the case of a LastArrival gate will be much higher as the reset signal must be forwarded to the input ports that have not received a data pulse yet.

When such a reset signal is used, the target gate may return to its initial state; however, in many cases an output pulse is generated too. This output pulse propagates through

[6] RSFQ logic gates have by default only one fanout.

[7] Globally asynchronous, locally synchronous (GALS) clocking is a commonly-used technique to mitigate timing variations across functional blocks and reduce clock tree overhead in CMOS as well [8].

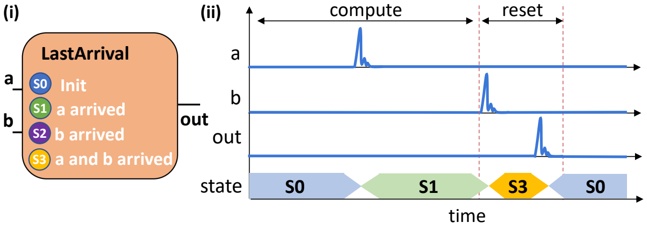


Figure 5. LastArrival can be in one of the four states shown in Panel (i). To reset this component both input pulses *a* and *b* must arrive. Panel (ii) shows how a spacer period can be used to avoid the interference between data pulses associated with the actual computation and pulses generated due to a reset signal.

the circuit in a downstream fashion and may affect the state of other subsequent gates. So, resetting a deep temporal circuit may have to be done sequentially – one stage of gates at a time.

To avoid the interference of data pulses that relate to the actual computation with the ones generated by resetting, a *spacer* period will also be needed. Figure 5 illustrates this scenario for a plain LastArrival gate. Once the "compute" period ends, a reset pulse is sent to its input port *b*. Any output pulse observed until the next compute period starts should be ignored. If the LastArrival gate is connected to other gates, the generated output pulse may also affect their state even if they have already been cleared. So, resetting in that case must happen step-by-step and the duration of the spacer period will have to be adjusted accordingly.

An alternative method that we can rely on for resetting is to adjust the amount of applied bias current; setting the applied bias current to zero will release the stored flux quanta and return the gates to their initial states. This solution does not require additional hardware and comes without the concerns related to the propagation of reset-generated pulse. However, it is still not "free".

Choosing between these two options depends on the structure of the constructed circuit, possible resource constraints, and the corresponding delay associated with each of these methods.

5 Evaluation

In the previous sections, a framework for understanding the proposed RSFQ-based temporal computing paradigm at the logic, primitive gate, and device-levels has been presented. We may now leverage this understanding to functionally validate our design methodology through a number of accelerator designs. For the timing and functional validation of the developed circuits, we first identify timing constraints that affect the design flow and then provide corresponding SPICE-level simulation results. Finally, we compare their performance against their CMOS counterparts, showing more than an order of magnitude improvements.

Experimental setup: We perform our analysis based on the open-source WRSPICE circuit simulator [13] using the MIT-LL SFQ5ee 10 kA/cm^2 process. For our designs' interconnections, we use JTLs along with Splitters ($\mathfrak{s}$) and Mergers ($\mathfrak{m}$) where required.

5.1 Timing Analysis

Computing based on temporal relationships is in many cases naturally immune to noise as the final outcome often depends on the interval or the order in which events occur and not precise arrival times. Under conventional binary encoding, an early or late pulse translates to a bit-flip and its effect on the computation's accuracy depends on the bit's position. Under delay representation though, a time-skewed pulse may or may not affect the encoded value – in reality, an interval rather than a specific time is used to represent a value – and that may not even change the rank order of the occurring events.

To ensure the robustness of our designs though we cannot rely solely on the properties of our encoding and temporal logic. Understanding the various timing constraints is critical for reasoning about our circuits' behavior and developing a systematic way for the design of temporal RSFQ accelerators. To address this concern, in the following, we first introduce the required terminology for our timing analysis and then proceed with the description of the timing constraints of temporal circuits and the quantification of our primitives' robustness to the timing skew of pulses.

Figure 6 provides an illustration of the main timing relationships between pulses in our architecture. Data-to-data (t_{D2D}) window represents the time difference between two input data signals. Clock to Q (t_{C2Q}) denotes the delay between clock signal arrival and the occurrence of the output event. The propagation delays of the Splitter and Merger are shown as t_s and t_m, respectively. Finally, t_{su} represents the setup time, which denotes the minimum amount of time required between the arrival of data and clock signals, while t_c is the time window where input pulses are forbidden to arrive [22].

To avoid setup time violations, t_{D2D} has to be less than $t_m - t_{su}$ if the two input pulses represent the same value. To increase this time window, delay elements can be added after the Merger. In the case where two input pulses represent two consecutive values (e.g. $din0 = 2$ and $din1 = 3$), t_{D2D} has to be greater than $t_m + t_c$. If t_{D2D} is smaller than $t_m + t_c$, either the second input pulse will get "lost" (timing violation) or both pulses will be considered to represent the same value (e.g. $din0 = 2$ and $din1 = 2$), which is incorrect.

Stretching the "valid" data time window of a cycle is possible with the use of additional JTLs; e.g., if we want the "valid"

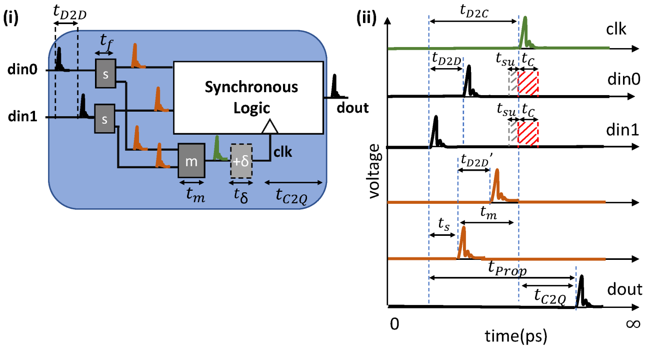


Figure 6. Illustration of the various timing constraints in the case of synchronous temporal RSFQ blocks.

data time window of a cycle to go from 10 ps to 20 ps, four rather than two JTLs have to be used for the realization of $\mathbf{D}1\phi$. Obviously, this change comes at the cost of area (more JTLs mean more JJs) and performance (each cycle will last longer); but, it results in a much smaller chance of a pulse getting lost in a synchronous component due to imprecision associated with variability or noise. To better understand and quantify the tolerance of our designs to time skew, we perform a number of detailed SPICE-level simulations. These simulations allow us to analyze the sensitivity of temporal gates to pulses under various t_{D2D}s.

As expected, imprecise pulses do not affect the correct operation of **FA**, **D**, and **LA**, which are implemented with Merge, JTL, and C-element components; all of these circuits are by nature clockless. The case of Inhibit is of particular interest though as although its clock signal has been repurposed, still the timing constraints described earlier apply.

Figure 7 provides simulation results of this case – $\psi \mathbf{I_s} \phi$ – for various t_{D2D}s: -5, 0, and 2 ps. As can be seen, if ψ arrives 5 ps before ϕ, the output of the Inhibit gate remains "0" (the correct value), while an output spike gets produced for t_{D2D} > -5 ps. So, if two input pulses representing two different values are always more than 5ps apart, Inhibit is guaranteed to work as expected. However, if two variables have the same value, $\mathbf{D}1x$ has to be greater than 5 ps and the data input pulse ϕ may need to be delayed – so it will appear towards the end of the assigned interval – in order to avoid the occurrence of an undesired output pulse.

For the verification of more complex designs under timing uncertainty, the formalism introduced in Section 3.2 can be used. Besides model checking, with the help of the proposed function $\mathcal{E}_{\langle S,t \rangle}()$ events occurrences can be translated to numbers and incorporated into an interval analysis. Such an analysis is beyond the scope of this paper; however, we foresee its potential for the reasoning of superconducting temporal designs in noisy settings, where understanding (and quantifying) how timing skews add up may be critical for the correct behavior and efficiency of the system.

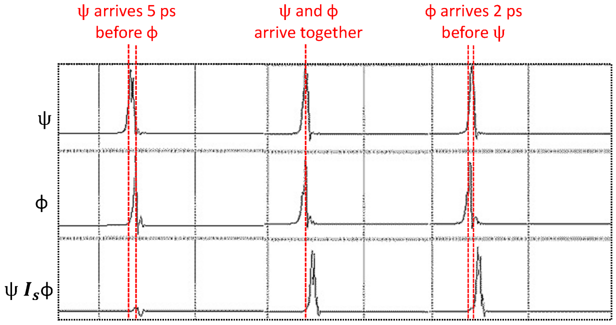


Figure 7. Timing analysis for the Inhibit gate.

5.2 Proof-of-Concept

As a proof-of-concept, we design and simulate temporal RSFQ accelerators for (a) DNA sequencing, (b) decision trees, and (c) arbitrary function tables. The temporal DNA sequencing algorithm was presented in the original race logic paper [19]. Both synchronous [19] and asynchronous [20] CMOS implementations have been demonstrated since then. "Race" decision trees [33] are another interesting application. Race trees demonstrate the utility of temporal logic to classification problems. For the realization of their decoders the use of a NOT gate is required; NOT is not one of temporal logic's primitives and its functionality in the temporal domain is different than in Boolean logic. Finally, in contrast to these two designs that are *purely* asynchronous, for the implementation of the circuit realizing an arbitrary function table [25] the use of both synchronous and asynchronous components is needed, providing a great opportunity to showcase the effectiveness of our data-driven self-timing scheme.

5.2.1 Needleman-Wunsch Sequence Alignment

Needleman and Wunsch's algorithm was one of the first applications of dynamic programming to compare biological sequences. The algorithm assigns a score to every possible alignment and its purpose is to find all possible alignments having the highest score. In more detail, the main idea behind this algorithm is that initially a 2D grid will be constructed out of two arbitrary strings P and Q – Figure 8 (i) – and then, for each individual pair of letters a score will be chosen; each operation – *deletion*, *insertion*, *match* – is associated with a different directed edge, where each edge can have its own score/weight.

In the algorithm's temporal realization, each score associates to a delay. Hence, the total time required for a single "pulse" to propagate from the array's input to the output reveals the desired similarity score. The architecture of this circuit can be generally thought of as a systolic array, where each cell is implemented in RSFQ, as shown in Figure 8 (iii); Figure 8 (ii) shows its CMOS implementation, as proposed by A. Madhavan, et al. [19]. The penalties for deletion, insertion, and match are in the shown example set to 1.

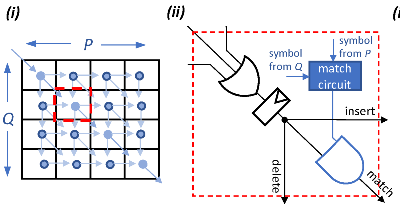


Figure 8. Panel (i): 2D grid constructed following Needleman and Wunsch's algorithm. Panel (ii): schematic of CMOS unit cell for the implementation of a DNA sequence alignment temporal accelerator. Panel (iii): RSFQ equivalent circuit. *P* and *Q* represent the two DNA strings to be aligned. If there is a "match" the match circuit returns *True*; otherwise, it returns *False*. The penalties/delays for deletion, insertion, and match are set to 1.

In contrast to the synchronous CMOS case, where the similarity score is incremented by one as the first arriving pulse goes through a flip-flop, in our asynchronous RSFQ implementation, D1x matches the propagation delay of each unit cell. Thus, every time a pulse goes through a unit cell the score will increment by one. To control the propagation of a pulse across the diagonal, which should happen only when a match occurs, a JJ is used. The switching operation is performed by changing the value of the bias current applied to the JJ; if the current is too low, an incoming RSFQ pulse cannot cause the JJ to fire, which allows for the CMOS control of the circuit.

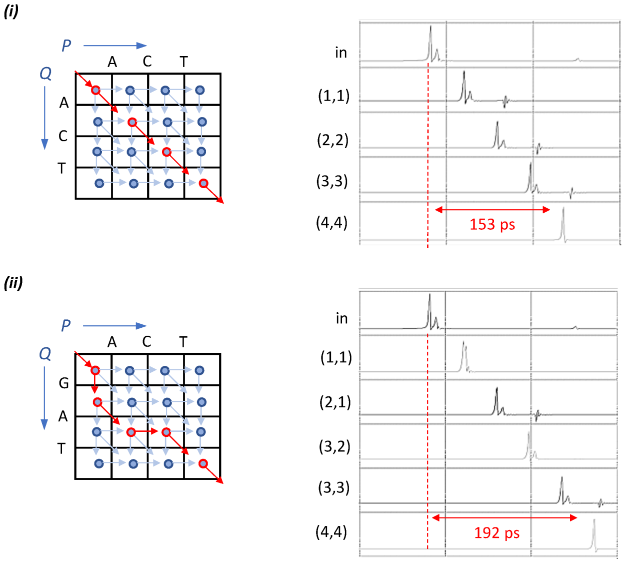


Figure 9. Shortest path and simulation results for a 3×3 DNA sequence alignment problem. Panel (i): *P* = ACT and *Q* = ACT. Panel (ii): *P* = ACT and *Q* = GAT.

Figure 9 shows WRSPICE simulation results for a 3×3 DNA sequence alignment problem. In Panel (i), *P* = ACT and *Q* = ACT are compared. Considering that the two strings perfectly match, the shortest path from the grid's input to output cell will be across its diagonal – consisting of four unit cells – and results in a delay of 153 ps. In Panel (ii), where the strings *P* = ACT and *Q* = GAT are compared, the propagation delay of a pulse across the grid is 192 ps; the shortest path now consists of five rather than four unit cells. These results match our expectations; in our experiments, the penalties for deletion, insertion, and match are set to 1 and correspond to a 38 ps delay – equal to the propagation delay of each unit cell.

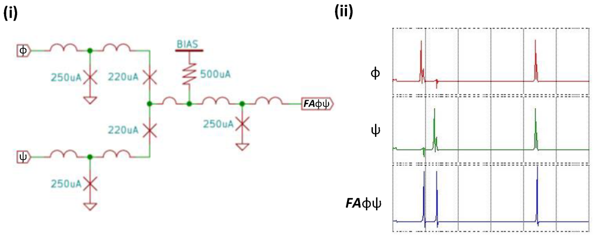


Figure 10. Panel (i): schematic of an RSFQ Merge element realizing a stateless FirstArrival gate. Panel (ii): WRSPICE simulation results.

In some cases, where race logic constraints can be safely relaxed, superconducting hardware can also provide a trade-off space that enables optimization for select parameters, such as area, latency, power consumption, and complexity (which can play a role in the susceptibility of the circuit to variability). One example of this can be employed in the sequencing accelerator, in which a stateless FirstArrival gate – composed of just a Merger, as depicted in Figure 10 – may be used as an alternative to the stateful version presented above. The outcome is an accelerator with fewer JJs (28 rather than 36 JJs are now required per unit cell) and a ∼14% lower latency. Corresponding simulation results for the two example cases discussed above are shown in Figure 11.

More performance results and a comparison between our RSFQ sequencing accelerators and their CMOS counterparts can be found in Table 3.

Table 3. Estimated (best and worst) latency results for DNA sequencing accelerator in both CMOS (0.5μm) [19] and RSFQ.

Strings Length	CMOS Latency	**RSFQ Latency w/ stateful FAs**	**Improv. w/ stateful FAs**	**RSFQ Latency w/ stateless FAs**	**Improv. w/ stateless FAs**
40	50 ns - 100 ns	1.6 ns - 3.1 ns	∼32×	1.4 ns - 2.7 ns	∼37×
60	75 ns - 150 ns	2.3 ns - 4.6 ns	∼32×	2 ns - 4 ns	∼37×
80	100 ns - 200 ns	3.1 ns - 6.2 ns	∼32×	2.7 ns - 5.4 ns	∼37×

It should be noted that while replacing one or more of our basic temporal RSFQ primitives with simpler ones may in

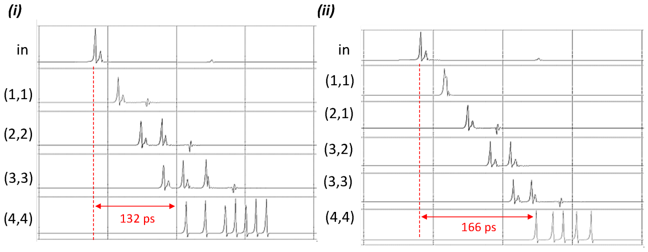


Figure 11. Simulation results for a stateless implementation of the sequencing accelerator depicted in Figures 8 and 9. Panel (i): P = ACT and Q = ACT. Panel (ii): P = ACT and Q = GAT.

some cases be appealing, it is not always safe. For example, when using a plain Merger as a FirstArrival gate, more than one output pulses may be generated, which violates race logic's constraint for the existence of at most one pulse per "wire". In the case of the sequencing accelerator, this "relaxation" does not cause any malfunction. However, if, for example, Coincidence or Inhibit gates followed a Merge-based FirstArrival gate then the possibility of an error exists; the first spike coming out of a FirstArrival or a Delay gate is always valid, this is not the case though for all gates. To verify whether such a replacement is safe or not the formalism introduced in Section 3.2 can be used.

5.2.2 Race Trees

An ensemble of decision trees can be implemented in race logic, as described by Tzimpragos, et al. [33]. In the case of *Race Trees*, each tree node can be considered an independent temporal threshold function and be realized with a single Inhibit operator.

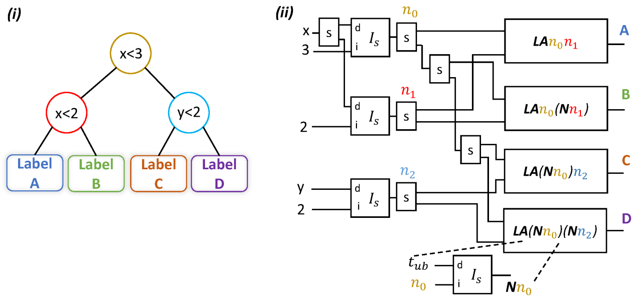


Figure 12. Panel (i): a decision tree with three nodes. Panel (ii): temporal RSFQ implementation of that tree.

Figure 12 (ii) shows the RSFQ equivalent of the CMOS implementation provided in the original paper – realizing the decision tree shown in Figure 12 (i). For the design of the "label" decoder, the use of a NOT gate is required. In contrast to Boolean logic though, NOT is not a primitive temporal operator. For its construction, we use Inhibit and an *upper bound* reference signal t_{ub}, which denotes the end of a specific time interval of interest (directly related to the inputs resolution in this case). Hence, NOT will fire at $t = t_{ub}$ if and only if the gate has received no input spikes from time reference 0 until that moment.

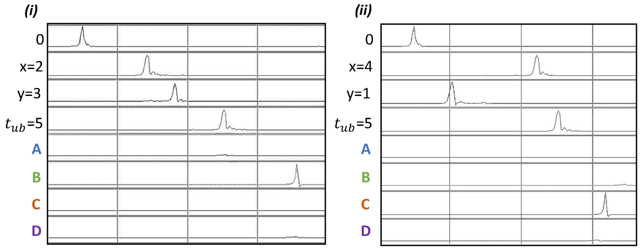


Figure 13. Panel (i): WRSPICE simulation results for $x = 2$, $y = 3$, and $t_{ub} = 5$. Panel (ii): WRSPICE simulation results for $x = 4$, $y = 1$, and $t_{ub} = 5$.

WRSPICE simulation results are provided in Figure 13. In the first case, inputs x and y are equal to 2 and 3, while in the second one, x and y are set to 4 and 1. Moreover, the upper bound reference signal t_{ub} is set to 5 and $\mathbf{D}1x$ corresponds to a 25 ps delay. Associating a smaller delay with $\mathbf{D}1x$ may be possible; however, our main goal in this paper is to demonstrate the correct functionality of the design rather than optimizing its performance. As expected, the final outcome is Label B for the former and Label C for the latter. The total latency is 150 ps and the design consists of 166 JJs.

More performance results and a comparison with its CMOS counterpart can be found in Table 4.

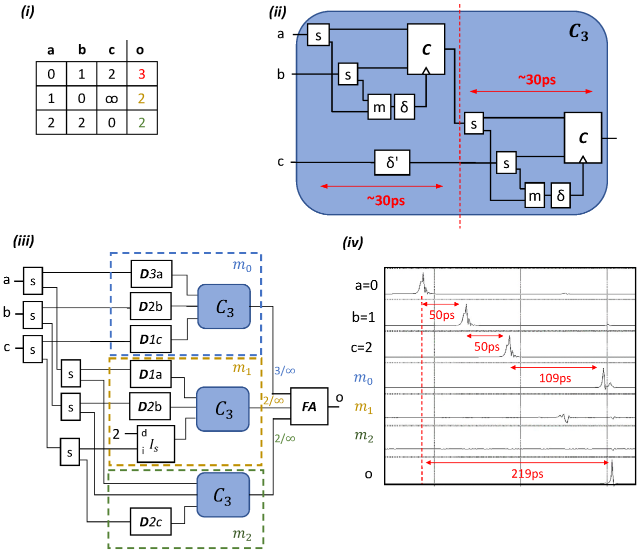
Table 4. Estimated latency results for hardwired Race Trees in both CMOS (f = *1* GHz) [33] and RSFQ ($\mathbf{D}1x$ = 25 ps).

# Trees	Depth	Inp. res.	CMOS Latency	**RSFQ Latency**	**Improvement**
1	6	4 bits	17 ns	0.464 ns	37×
1	6	8 bits	257 ns	6.464 ns	40×
1	8	4 bits	17 ns	0.490 ns	35×
1	8	8 bits	257 ns	6.490 ns	40×

5.2.3 Arbitrary Function Table

Finally, we implement in RSFQ the feedforward temporal network previously presented by J. E. Smith [25].

Figure 14 (i) provides the specification of our example feedforward temporal network. Considering that our function has three input variables and given the limited fan-in of our basic operators, the main building block C_3 (shown in Figure 14 (ii)) of our architecture (shown in Figure 14 (iii)) will consist of two 2-ary Coincidence gates; in contrast to the original design, where Coincidence consists of LastArrival, FirstArrival, Inhibit, and Delay gates, we opt for the AND gate-based implementation described in Section 4.1.



a	b	c	o
0	1	2	3
1	0	∞	2
2	2	0	2

Figure 14. Panel (i): specification of an example function table. Panel (ii): Block diagram of a self-timed 3-input Coincidence gate . Panel (iii): Block diagram showing our accelerator's architecture. Panel (iv): WRSPICE simulation results for $a = 0$, $b = 1$, and $c = 2$.

For its clocking, we apply the data-driven self-timing scheme proposed in Section 4.2. To successfully handle time-skewed inputs a delay $\delta = 10$ ps is introduced after each Merger. A delay element δ' is also used to balance the delays of the two parallel paths that feed the second Coincidence gate.

Simulation results can be found in Figure 14 (iv). In our simulation, D$1x$ is set to 50 ps and the inputs provided are $a = 0$, $b = 1$, and $c = 2$. As expected, a spike will appear at the output of the upper block m_0, colored in red, at $t = 209$ ps and will go through the succeeding 3-input FirstArrival gate (stateless rather than stateful FirstArrival gates are used again); the propagation delay of C_3 is 60 ps , so if we subtract that from the total delay we will end up with 149 ps of delay which corresponds to the desired value 3. No spikes will come out of the other two "blocks", colored in blue and green, corresponding to two bottom entries of the function table. Our circuit design consists of 565 JJs and its latency is 219 ps.

6 Conclusion

Superconducting SFQ technologies are a promising candidate for high-speed and ultralow-energy operation for certain classes of computation. Though both the underlying physics and basic circuit technologies are well understood, many hurdles remain before larger computations can enjoy the benefits of superconducting materials.

While some of the challenges ahead are fundamental to the device physics of superconducting, it is also important to realize that the traditional logic abstractions and digital design patterns we understand so well have co-evolved with the hardware technology that has embodied them. As we look past CMOS, there is no reason to think that those same abstractions best serve to encapsulate the computational potential inherent to emerging devices. Computational efficiency is always lost through abstraction, yet successful abstractions will keep the most useful aspects of a system while simultaneously enabling composition, scale, optimization, and verification.

In this paper, we demonstrate a new foundation that bridges the gap between the level-driven logic traditional hardware designs accept as a foundation and the pulse-driven logic naturally supported by the most compelling superconducting technologies. The key to this new foundation is the harmonious interaction between three different areas of work – superconducting logic, temporal predicate logic, and delay-based codes. We show that superconducting logic can naturally compute over temporal relationship between pulse arrivals, we formalize and provide implementation circuits for fundamental operators in temporal logic, we propose an asynchronous data-driven self-timing scheme, and we perform a timing analysis to identify timing constraints that affect the design flow of superconducting temporal accelerators. Finally, to validate our hypothesis we implement three temporal accelerators in RSFQ and compare their performance against their CMOS counterparts, showing more than an order of magnitude improvements.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 1740352, 1730309, 1717779, 1563935, and a gift from Cisco Systems.

Research carried out at LBNL is operated for the U.S. Department of Energy Office of Science under Contract No. DE-AC02-05CH11231.

Dr. Madhavan also acknowledges support under the Cooperative Research Agreement between the University of Maryland and the National Institute of Standards and Technology Physical Measurement Laboratory, Award 70NANB14H209, through the University of Maryland.

Last but not least, the authors would like to thank James E. Smith, David Donofrio, Dmitri Strukov, Alexander Wynn, and the anonymous reviewers for their helpful comments.

References

[1] Y. Ando, R. Sato, M. Tanaka, K. Takagi, N. Takagi, and A. Fujimaki. 2016. Design and Demonstration of an 8-bit Bit-Serial RSFQ Microprocessor: CORE e4. *IEEE Transactions on Applied Superconductivity* 26, 5 (Aug 2016), 1–5. https://doi.org/10.1109/TASC.2016.2565609

[2] D. A. Buck. 1956. The Cryotron-A Superconductive Computer Component. *Proceedings of the IRE* 44, 4 (April 1956), 482–493. https://doi.org/10.1109/JRPROC.1956.274927

[3] 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 (ISCA '19)*. ACM, New York, NY, USA, 567–578. https://doi.org/10.1145/3307650.3322270

[4] W. Chen, A. V. Rylyakov, V. Patel, J. E. Lukens, and K. K. Likharev. 1999. Rapid single flux quantum T-flip flop operating up to 770 GHz. *IEEE Transactions on Applied Superconductivity* 9, 2 (June 1999), 3212–3215. https://doi.org/10.1109/77.783712

[5] Alessandro Cimatti, Marco Roveri, and Daniel Sheridan. 2004. Bounded Verification of Past LTL. In *Formal Methods in Computer-Aided Design*, Alan J. Hu and Andrew K. Martin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 245–259.

[6] Z. J. Deng, N. Yoshikawa, S. R. Whiteley, and T. Van Duzer. 1997. Data-driven self-timed RSFQ digital integrated circuit and system. *IEEE Transactions on Applied Superconductivity* 7, 2 (June 1997), 3634–3637. https://doi.org/10.1109/77.622205

[7] M. Dorojevets, P. Bunyk, and D. Zinoviev. 2001. FLUX chip: design of a 20-GHz 16-bit ultrapipelined RSFQ processor prototype based on 1.75-/spl mu/m LTS technology. *IEEE Transactions on Applied Superconductivity* 11, 1 (March 2001), 326–332. https://doi.org/10.1109/77.919349

[8] Eby G. Friedman. 1997. *High Performance Clock Distribution Networks*. Springer US, Boston, MA, 1–4. https://doi.org/10.1007/978-1-4684-8440-3_1

[9] Dov Gabbay. 1989. The declarative past and imperative future. In *Temporal Logic in Specification*, B. Banieqbal, H. Barringer, and A. Pnueli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 409–448.

[10] Kris Gaj, Eby G. Friedman, and Marc J. Feldman. 1997. Timing of Multi-Gigahertz Rapid Single Flux Quantum Digital Circuits. *Journal of VLSI signal processing systems for signal, image and video technology* 16, 2 (01 Jun 1997), 247–276. https://doi.org/10.1023/A:1007903527533

[11] D. S. Holmes, A. M. Kadin, and M. W. Johnson. 2015. Superconducting Computing in Large-Scale Hybrid Systems. *Computer* 48, 12 (Dec 2015), 34–42. https://doi.org/10.1109/MC.2015.375

[12] D. S. Holmes, A. L. Ripple, and M. A. Manheimer. 2013. Energy-Efficient Superconducting Computing-Power Budgets and Requirements. *IEEE Transactions on Applied Superconductivity* 23, 3 (June 2013), 1701610–1701610. https://doi.org/10.1109/TASC.2013.2244634

[13] Whiteley Research Incorporated. 2019. *WRspice reference manual.* Technical Report. http://www.wrcad.com/manual/wrsmanual.pdf

[14] H. Kamerlingh Onnes. 1911. The resistance of pure mercury at helium temperatures. *Commun. Phys. Lab. Univ. Leiden, b* 120 (1911).

[15] Hans Kamp. 1968. *Tense Logic and the Theory of Linear Order.* Ph.D. Dissertation. Ucla.

[16] N. K. Katam, J. Kawa, and M. 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

[17] Konstantin K. Likharev. 2012. Superconductor digital electronics. *Physica C: Superconductivity and its Applications* 482 (2012), 6 – 18. https://doi.org/10.1016/j.physc.2012.05.016 2011 Centennial superconductivity conference - EUCAS-ISEC-ICMC.

[18] K. K. Likharev and V. K. Semenov. 1991. RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Transactions on Applied Superconductivity* 1, 1 (March 1991), 3–28. https://doi.org/10.1109/77.80745

[19] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. 2014. Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms. *SIGARCH Comput. Archit. News* 42, 3 (June 2014), 517–528. https://doi.org/10.1145/2678373.2665747

[20] A. Madhavan, T. Sherwood, and D. Strukov. 2017. A 4-mm2 180-nm-CMOS 15-Giga-cell-updates-per-second DNA sequence alignment engine based on asynchronous race conditions. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*. 1–4. https://doi.org/10.1109/CICC.2017.7993630

[21] R. Manohar. 2015. Comparing Stochastic and Deterministic Computing. *IEEE Computer Architecture Letters* 14, 2 (July 2015), 119–122. https://doi.org/10.1109/LCA.2015.2412553

[22] O. A. Mukhanov, S. V. Rylov, V. K. Semonov, and S. V. Vyshenskii. 1989. RSFQ logic arithmetic. *IEEE Transactions on Magnetics* 25, 2 (March 1989), 857–860. https://doi.org/10.1109/20.92421

[23] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan. 2018. Low-Cost Sorting Network Circuits Using Unary Processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 8 (Aug 2018), 1471–1480. https://doi.org/10.1109/TVLSI.2018.2822300

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

[25] James E. Smith. 2018. Space-Time Algebra: A Model for Neocortical Computation. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA '18)*. IEEE Press, 289–300. https://doi.org/10.1109/ISCA.2018.00033

[26] James E. Smith. 2019. (Newtonian) Space-Time Algebra. arXiv:cs.LO/2001.04242

[27] Igor I Soloviev, Nikolay V Klenov, Sergey V Bakurskiy, Mikhail Yu Kupriyanov, Alexander L Gudkov, and Anatoli S Sidorenko. 2017. Beyond Moore's technologies: operation principles of a superconductor alternative. *Beilstein Journal of Nanotechnology* 8 (Dec 2017), 2689–2710. https://doi.org/10.3762/bjnano.8.269

[28] M. Tanaka, R. Sato, Y. Hatanaka, and A. Fujimaki. 2016. High-Density Shift-Register-Based Rapid Single-Flux-Quantum Memory System for Bit-Serial Microprocessors. *IEEE Transactions on Applied Superconductivity* 26, 5 (Aug 2016), 1–5. https://doi.org/10.1109/TASC.2016.2555905

[29] G. Tang, P. Qu, X. Ye, and D. Fan. 2018. Logic Design of a 16-bit Bit-Slice Arithmetic Logic Unit for 32-/64-bit RSFQ Microprocessors. *IEEE Transactions on Applied Superconductivity* 28, 4 (June 2018), 1–5. https://doi.org/10.1109/TASC.2018.2799994

[30] G. Tang, K. Takata, M. Tanaka, A. Fujimaki, K. Takagi, and N. Takagi. 2016. 4-bit Bit-Slice Arithmetic Logic Unit for 32-bit RSFQ Microprocessors. *IEEE Transactions on Applied Superconductivity* 26, 1 (Jan 2016), 1–6. https://doi.org/10.1109/TASC.2015.2507125

[31] Swamit S. Tannu, Poulami Das, Michael L. Lewis, Robert Krick, Douglas M. Carmean, and Moinuddin K. Qureshi. 2019. A Case for Superconducting Accelerators. In *Proceedings of the 16th ACM International Conference on Computing Frontiers (CF '19)*. ACM, New York, NY, USA, 67–75. https://doi.org/10.1145/3310273.3321561

[32] Sergey K Tolpygo. 2016. Superconductor digital electronics: Scalability and energy efficiency issues. *Low Temperature Physics* 42, 5 (2016), 361–379.

[33] Georgios Tzimpragos, Advait Madhavan, Dilip Vasudevan, Dmitri Strukov, and Timothy Sherwood. 2019. Boosted Race Trees for Low Energy Classification. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 215–228. https://doi.org/10.1145/3297858.3304036

[34] Y. Yamanashi, M. Tanaka, A. Akimoto, H. Park, Y. Kamiya, N. Irie, N. Yoshikawa, A. Fujimaki, H. Terai, and Y. Hashimoto. 2007. Design and Implementation of a Pipelined Bit-Serial SFQ Microprocessor, CORE1β. *IEEE Transactions on Applied Superconductivity* 17, 2 (June 2007), 474–477. https://doi.org/10.1109/TASC.2007.898606