University of California
Santa Barbara

Computing with Temporal Operators

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Georgios Tzimpragos

Committee in charge:

Professor Timothy Sherwood, Chair
Professor Yufei Ding
Professor Yu Feng
Professor James E. Smith

September 2022

The Dissertation of Georgios Tzimpragos is approved.

Professor Yufei Ding

Professor Yu Feng

Professor James E. Smith

Professor Timothy Sherwood, Committee Chair

June 2022

Computing with Temporal Operators

To my family and friends, for making my days brighter.

Acknowledgements

This dissertation would not have been possible without the support and contributions of many people. First, I want to thank my advisor, Tim Sherwood, for setting up an environment where I could be myself and feel free to explore my most wacky ideas. His truly interdisciplinary research approach opened a new and fascinating world for me, and I could not be more grateful for this.

Next, I want to thank all the UC Santa Barbara Computer Science Department faculty and staff members for their inspiration and help over the years. Moreover, David Donofrio and George Michelogiannakis deserve credit for believing in my research and supporting my first superconducting endeavors. The extended technical discussions with Scott Holmes contributed to taking these efforts to the next level too. At this point, I also want to acknowledge Amit Majumdar for his patience and encouragement, Dmitri Strukov for his support and feedback, and all the members of NTUA's μlab for introducing me to research and always being there to listen to my latest adventures.

I am also extremely thankful to my close collaborators: Jennifer Volk, Alex Wynn, Evan Golden, and Michael Christensen. Working with them broadens my horizons and triggers my curiosity in unexpected and fun ways. Their positive attitude and support also significantly impact me, and I am lucky to have met them.

Last but not least, I want to acknowledge Jim Smith. Jim is not only a true inspiration as a scientist but also a fantastic human. His determination, energy, and free spirit give me optimism and motivate me to put my heart into whatever I do. Thank you, Jim.

Curriculum Vitæ
Georgios Tzimpragos

Education

2022	Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2016	M.S. in Electrical and Computer Engineering, University of California, Davis.
2012	B.S. in Electrical and Computer Engineering, National Technical University of Athens.

Publications

G. Tzimpragos, J. Volk, A. Wynn, T. Sherwood. Pulsar: A Superconducting Delay-Line Memory. *arXiv:2205.08016.* 2022.

M. Christensen, **G. Tzimpragos**, H. Kringen, J. Volk, T. Sherwood, B. Hardekopf. PyLSE: A Pulse-Transfer Language for Superconductor Electronics. *ACM 43rd Conference on Programming Language Design and Implementation (PLDI).* 2022.

G. Tzimpragos, J. Volk, A. Wynn, J. E. Smith, T. Sherwood. Superconducting Computing with Alternating Logic Elements. *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA).* 2021.

G. Tzimpragos, A. Madhavan, D. Vasudevan, D. Strukov, T. Sherwood. In-sensor Classification with Boosted Race Trees. *Commun. ACM 64, 6.* 2021.

Y. Wang, B. Feng, G. Li, **G. Tzimpragos**, L. Deng, Y. Xie, Y. Ding. TiAcc: Triangle Inequality-Based Hardware Accelerator for K-Means on FPGAs. *ACM/IEEE 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid).* 2021.

G. Tzimpragos, J. Volk, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Shalf, T. Sherwood. Temporal Computing With Superconductors. *IEEE Micro, vol. 41, no. 3.* 2021.

A. Majumdar, **G. Tzimpragos**, J. Villarreal, K. Deepak, J. Rangarajan. Breakpointing Circuitry That Evaluates Breakpoint Conditions While Running Clock to Target Circuit. US 10,754,759 B1. 2020.

D. Dangwal, **G. Tzimpragos**, T. Sherwood. Agile Hardware Development and Instrumentation with PyRTL. *IEEE Micro, vol. 40, no. 4.* 2020.

G. Tzimpragos, J. Villarreal, A. Majumdar, K. Deepak, Y. Zhu. Data Unit Breakpointing Circuits and Methods. US 10,621,067 B1. 2020.

G. Tzimpragos, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Volk, J. Shalf, T. Sherwood. A Computational Temporal Logic for Superconducting

Accelerators. *ACM 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2020.

W. Cui, **G. Tzimpragos**, Y. Tao, J. McMahan, D. Dangwal, N. Tsiskaridze, G. Michelogiannakis, D. Vasudevan, T. Sherwood. Language Support for Navigating Architecture Design in Closed Form. *J. Emerg. Technol. Comput. Syst. (JETC) 16, 1, Article 9.* 2020.

G. Tzimpragos, A. Madhavan, D. Vasudevan, D. Strukov, T. Sherwood. Boosted Race Trees for Low-Energy Classification. *ACM 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2019.

W. Cui, Y. Ding, D. Dangwal, A. Holmes, J. McMahan, A. Jabari-Abhari, **G. Tzimpragos**, F. Chong, T. Sherwood. Charm: A Language for Closed-Form High-Level Architecture Modeling. *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 2018.

J. Clow, **G. Tzimpragos**, D. Dangwal, S. Guo, J. McMahan, T. Sherwood. A Pythonic Approach for Rapid Hardware Prototyping and Instrumentation. *27th International Conference on Field Programmable Logic and Applications (FPL)*. 2017.

G. Tzimpragos, D. Cheng, S. Tapp, B. Jayadev, A. Majumdar. Application Debug in FPGAs in the Presence of Multiple Asynchronous Clocks. *International Conference on Field-Programmable Technology (FPT)*. 2016.

G. Tzimpragos, C. Kachris, I. B. Djordjevic, M. Cvijetic, D. Soudris, I. Tomkos. A Survey on FEC Codes for 100G and Beyond Optical Networks. *IEEE Communications Surveys & Tutorials, vol. 18, no. 1.* 2016.

R. Proietii, C. J. Nitta, Z. Cao, M. Clements, **G. Tzimpragos**, S. J. B. Yoo. Flexible-Bandwidth Power-Aware Optical Interconnects with Source-Synchronous Techniques. *IEEE Optical Interconnects Conference (OI)*. 2015.

C. Kachris, **G. Tzimpragos**, D. Soudris, I. Tomkos. Reconfigurable FEC Codes for Software-Defined Optical Transceivers. *13th International Conference on Optical Communications and Networks (ICOCN)*. 2014.

G. Tzimpragos, C. Kachris, D. Soudris, I. Tomkos. A Low-Complexity Implementation of QC-LDPC Encoder in Reconfigurable Logic. *23rd International Conference on Field programmable Logic and Applications (FPL)*. 2013.

Abstract

Computing with Temporal Operators

by

Georgios Tzimpragos

Binary codes and Boolean logic form the foundation of digital computing as we know it. However, the ever-increasing demand for cheap computing power for emerging applications and the advent of novel devices with unique characteristics bring about questions of potential alternatives. This dissertation challenges the status quo by reimagining the established digital/analog boundary and demonstrating how computing with temporal operators can be practical and remarkably efficient.

To this end, the focus is initially on conventional devices. The exploration begins with the idea that digital temporal codes, in which a number is represented by the time that a low to high voltage transition occurs, may, in some cases, be a happy medium between analog and digital binary. To showcase the benefits of this approach, a temporal accelerator for decision trees is developed. The resulting system is built solely with off-the-shelf CMOS components, allows tight integration with sensors, and delivers multiple orders of magnitude energy and performance gains over state-of-the-art solutions.

In the second part of the dissertation, the focus is on post-Moore technologies—specifically, superconductor electronics. Despite their promise as candidates for integrated classical-quantum computers and supercomputers, a fundamental mismatch between traditional computational abstractions and the pulse-based nature of superconductor devices impedes their advancement. Unfortunately, transient voltage pulses do not translate to 0s and 1s as easy as stable voltage levels. Fortunately, temporal operators do not use binary inputs. The advantages of avoiding pulse-to-binary translations at the gate level are highlighted through a series of temporal superconductor designs. Interestingly, these advantages carry over to Boolean superconductor designs by leveraging a newfound duality between temporal and Boolean operators.

The dissertation concludes with a discussion of superconducting information storage in the time domain and the generalization of temporal formalism in a way that allows the specification of both hardware and its properties using the same temporal operators.

Contents

Chapter 1

Overview

Complementary metal-oxide semincoductor (CMOS) scaling has been a driving factor for computing technology for many decades and the transistors manufactured today are multiple orders of magnitude more efficient, compact, and less expensive than those built 30 years ago. However, as CMOS approaches its limits, keeping this trend going becomes increasingly more difficult. This reality highlights the importance of cross-stack solutions that better utilize the room left at the top of the computing stack to get more out of existing devices or open up new opportunities for emerging ones.

A closer look at the top indicates the dependence on software engineering, algorithms, and hardware architectures for performance and efficiency improvements in the post-Moore era [1]. In the case of hardware architectures, a recent visionary paper [2] goes one step further and discusses the potential of five key topics that are expected to significantly change the computing field within the next 10 to 15 years: democratization of hardware specialization, architectural innovations for cloud computing, deep 3D integration, computing closer to physics, and machine learning as a key workload (see Figure 1.1).

The focus of this dissertation is primarily on machine learning as a key workload—

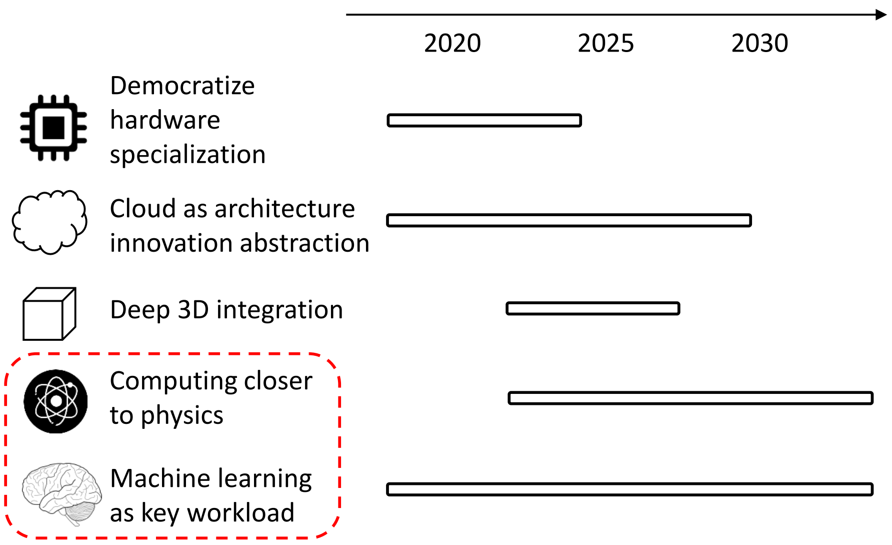


Figure 1.1: A vision of computer architecture research over the next decade, as described by Ceze, Hill, and Wenish [2]. The focus of this dissertation is outlined in red.

specifically, in-sensor classification (Chapter 2)—and computing closer to physics with superconductor devices (Chapters 3 and 4). The overarching goal is to show the benefits of customization at the data representation and logic levels, with temporal codes and operators being the center of attention here.

Thesis statement: With appropriate logic extensions, algorithms, and architecture techniques, computing with temporal operators can be practical and remarkably efficient both in established and emerging technologies.

1.1 Dissertation Organization and Contributions

The dissertation consists of three main parts. Briefly, the first part consists of Chapter 2 and focuses on temporal machine learning with CMOS devices. The second part consists of Chapters 3 and 4 and focuses on superconductor digital computing with temporal operators. The third and final part consists of Chapter 5, which looks into the future by approaching temporal computing through the lens of storage and verification. Consequently, Chapter 5 consists of two distinct sections. The first section presents a delay-line memory based on passive transmission lines with high kinetic inductance and unveils a new away of attacking the superconductor memory bottleneck. The second section draws a connection between propositional and computational temporal logic and shows a way to eliminate semantic gaps between hardware designs and temporal assertions that are commonly used in the context of formal verification. A more detailed summary of these contributions, organized in four thematic areas, is below.

1.1.1 Temporal In-Sensor Classification

Traditionally, engineers use time as the basis for important metrics and properties. For example, in general-purpose computing, faster is better, and execution time is a crucial performance metric [3].At the same time, in cyber-physical systems, actions must be taken at the correct time, and finishing early is no better than finishing late [4]. In contrast to these cases, the key idea behind my efforts is to use the passage of time as a computational resource and encode values in relative delays. This new perspective [5, 6] creates unique ways to work with sensors, especially those that return temporal information directly. For instance, temporal encoding subverts the costly time-to-digital conversion that typically happens in the output of dynamic vision and time-of-flight systems. However, temporal signals are fundamentally different than the binary inputs that Boolean operators expect. Instead of AND, OR, and NOT, the operators that I use to perform delay manipulations are those of First Arrival (FA), Last Arrival (LA), Delay (D), and Inhibit (I). Regarding machine learning, decision tree ensembles map well to these temporal operators at the algorithmic level, and the seamless cooperation of Boolean and temporal designs, implemented with off-the-shelf CMOS components, is demonstrated. The resulting architecture is fully programmable, has a shallow critical path, and minimizes switching. Finally, a development flow that accommodates this temporal logic scheme and integrates fully into the popular scikit-learn framework [7] is provided at the user interface level. The obtained results indicate that the proposed change is beneficial across all compute layers, from logic and circuit levels to application and integration.

1.1.2 Superconductor Logic Design with Temporal Operators

Pulse-based superconductor electronics (SCE) exhibit zero static power dissipation, speed-of-light energy-efficient interconnects, and clock rates in the 10s or 100s of GHz. However, a pulse, unlike a stable voltage level, is transient. This implies that superconductor cells must be stateful to perform any logical operation. But stateful cells need a mechanism to logically evaluate and reset. The conventional approach is to use a periodic clock signal. Unfortunately, the use of clocks in such a way leads to a variety of challenges across the entire hardware stack. I posit that temporal operators provide a better alternative to their Boolean counterparts by removing the need for clocking at the individual cell level [8, 9]. To validate this hypothesis, I first model First Arrival (FA), Last Arrival (LA), Delay, and Inhibit as Mealy machines. Then, I implement these Mealy machines in single flux quantum (SFQ) technology asynchronously and propose a methodology to combine them into larger superconducting accelerator architectures. Moreover, I discover a duality between Boolean and temporal operators that allows the repurposing of FA and LA SFQ cells as Boolean OR and AND functions [10]. To guarantee logical completeness and deterministically reinitialize the circuit state without the need for external signals, I use the theories of unordered codes and alternating logic. My simulation and analytical results indicate well over 10× performance and energy gains compared to conventional approaches for superconductors, with additional improvements expected at larger scales. To aid such explorations, an open-source pulse-transfer level language based on transition systems and timed automata is also developed [11].

1.1.3 Superconductor Delay-Line Memory

The logic contributions discussed above remove the clock from superconductor gate semantics and make superconductor electronics subject to the same techniques and methods from CMOS. However, a key component that is critical to any computer system is still missing: memory. The majority of prominent superconductor memory solutions envision a grid-like memory structure, which has promoted a focus on the miniaturization of memory cells. However, despite their promises, data densities remain unsatisfactorily low. This dissertation takes a different direction by using time again as a resource and presents a superconductor delay-line memory based on passive transmission lines with high kinetic inductance [12]. The developed memory system is fully superconducting, requires minimum control circuitry and fan-out, supports both sequential and content addressing, and operates at speeds ranging from 20 GHz to 100 GHz with sufficiently wide bias margins. The estimated densities surpass the state of the art by multiple orders of magnitude.

1.1.4 Formalism for Reasoning about Temporal Designs

The interplay between computing and time is multidimensional. A perspective that has not been discussed so far is that of formal temporal reasoning. Linear temporal logic was introduced for the first time in the late 1970s [13] as a method to specify and verify the properties of reactive systems. However, linear temporal logic (and its variants) is propositional and lacks the notion of "when", which signifies the moment in which a specific event happens and is critical for the temporal paradigm presented in this dissertation. To capture this missing notion, I extend the classical propositional temporal logic to a computational temporal logic capable of formally expressing delay-based computations [8, 9].

1.2 Permissions and Attributions

1. The text of Chapter 2 is in part a reprint of the material as it appears in the proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) and in the Communications of the ACM (CACM) journal, Vol. 64 No. 6. The dissertation author was the primary researcher. All the co-authors listed in these publications [5, 6] supervised or assisted in the research that forms the basis for Chapter 2.

2. The text of Chapter 3 is in part a reprint of the material as it appears in the proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), in the IEEE Micro journal (vol. 41, no. 3), and in the proceedings of the 48th International Symposium on Computer Architecture (ISCA). The dissertation author was the primary researcher. All the co-authors listed in these publications [8, 9] supervised or assisted in the research that forms the basis for Chapter 3.

3. The text of Chapter 4 is in part a reprint of the material as it appears in the proceedings of the 48th International Symposium on Computer Architecture (ISCA). The dissertation author was the primary researcher. All the co-authors listed in this publications [10] supervised or assisted in the research that forms the basis for

Chapter 4. The PyLSE language [11], used for Python-level simulations, was the result of a collaboration with Michael Christensen.

4. The text of Chapter 5 is in part a reprint of the material as it appears in the proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) and in arXiv:2205.08016. The dissertation author was the primary researcher. Jennifer Volk contributed equally to the superconductor delay-line memory research [12]. All the co-authors listed in these publications [8, 12] supervised or assisted in the research that forms the basis for Chapter 5.

   The pronoun used in the rest of the dissertation is *we* in recognition of my collaborators' contributions.

Chapter 2

Temporal In-Sensor Classification

2.1 Introduction

In embedded applications, where the computation and sensing are close in both time and space, the form of data is something that needs to be carefully considered. Typically, a sensor gathers analog information from the physical world and converts it into a conventional digital signal. For example, a camera captures incident photons and, through the photoelectric effect, uses their energy to guide cell charging The voltage on the cell is read out to an analog-to-digital converter (ADC) that transforms the measured voltage into a stream of zeros and ones. While this binary-represented integer is perfectly efficient for storage as bits in a memory and for general-purpose computing, it is unclear whether this is always the most energy-efficient solution. We posit that there are other encodings that are more efficient for in-sensor processing while still able to capture the relative values of the data to be encoded.

One such possible representation is pure analog signalling. While pure analog design promises power and performance gains, it comes with a number of its own challenges. First, well-understood analog design rules lag far behind digital rules in technology node.

High-density, high-performance, low-energy CMOS analog parts can be hard to achieve because of this gap. Second, while analog design in these aggressive technology nodes is certainly possible, tighter margins for process variations and noise often drive analog designs to use larger gates than their digital counterparts. Ideally, we could keep the good parts of analog behavior, where the computation closely matches the capabilities of the underlying devices, while keeping the noise tolerance and layout simplicity of digital designs.

One class of logic that attempts to strike this balance is race logic. The key idea behind race logic is to encode values as a delay from some reference [14]. All signals, unlike pure analog approaches, are supposed to be 0 or 1 at all times, with the time at which $0 \rightarrow 1$ transition happens encoding the value. Computations are then based on the relative propagation times of signals injected into a configurable circuit. The functions that form the base of race logic are: First Arrival (FA), Last Arrival (LA), Delay (D), and Inhibit (I). In prior work, First Arrival, Last Arrival, and Delay were used to build genome sequencing accelerators [14]. The inclusion of Inhibit provided completeness and opened the door to new computations [15]. However, the efficiency of this temporal approach to computing on larger and more general problems remains an open question.

To exploit the interesting new capabilities that this paradigm provides, we propose its application to a sensor-friendly and machine-learning-ready encoding. For the experimental validation of this hypothesis, we complete an end-to-end evaluation that includes estimates for energy, throughput, and area utilization in an ASIC (application-specific integrated circuit) design, a fully functional RTL (register-transfer level) implementation working in both simulation and on FPGA (field-programmable gate array), a fully automated toolchain linking scikit-learn [7] software structures down to device configurations, and an accuracy versus energy analysis across a set of decision tree ensembles and design parameters. Even excluding the energy savings at the interface level, the presented system dramatically reduces the total energy and latency required for in-sensor classification.

2.2 Generalized Race Logic

Race logic encodes values as relative delays. Computation then may happen through the purposeful manipulation of those delays rather than final logic levels, with LA, FA, D, and I forming the foundation of this logic; instead of the Boolean AND, OR, and NOT.

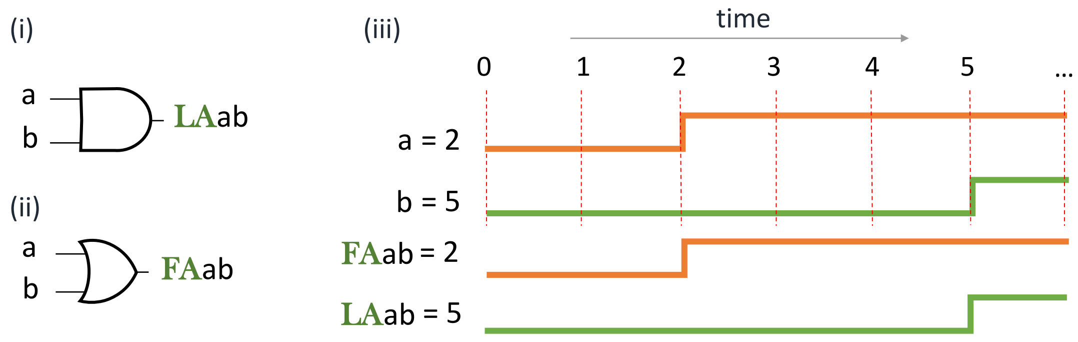


Figure 2.1: Panels (i) and (ii) show the implementation of Last Arrival (LA) and First Arrival (FA) functions with AND and OR gates. Panel (iii) represents an example waveform for $a = 2$ and $b = 5$.

As its name implies, LA outputs a high signal when all of its inputs have arrived. Its implementation in CMOS is simply an AND gate between its input wires, as an AND gate requires all its inputs to be high to fire an output. Similarly, all needed for a FA is an OR gate, as an OR gate outputs a high signal as soon as the first high input arrives. Figure 2.1 provides a waveform representation of LA's and FA's functionality.

Under the assumption that shorter delays encode smaller values and longer delays encode larger values, LA and FA can be thought of as max and min functions, respectively.

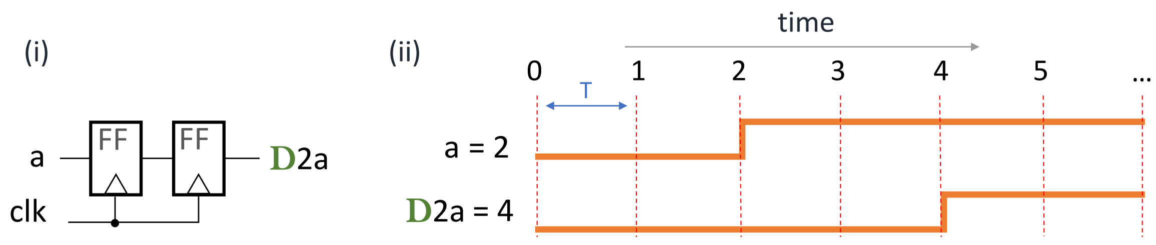


Figure 2.2: Under race logic, delaying (D) a $0 \rightarrow 1$ transition by k units of time can be thought of as a constant addition. Panel (i) shows how this delay can be achieved in conventional synchronous digital logic with the use of a shift-register. Panel (ii) shows an example waveform for $a = 2$ and $k = 2$.

This assumption also defines additions to be equivalent to delayed signal transitions. Delaying a signal by a fixed amount of time can be performed in multiple ways, depending on the implementation. In conventional synchronous digital logic, a sequence of flip-flops is sufficient, as shown in Figure 2.2. Asynchronous delay elements constructed out of current-starved inverters have also been demonstrated as a more energy-efficient alternative [16].

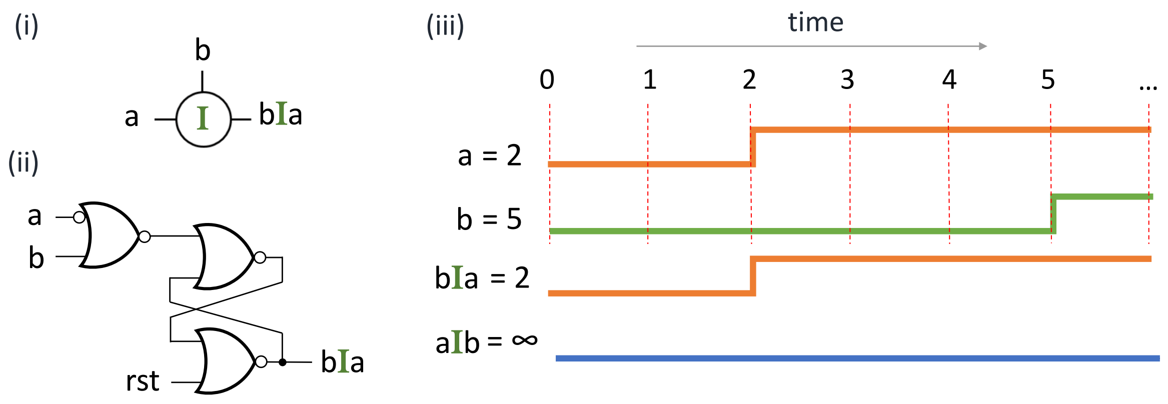


Figure 2.3: Panel (i) introduces the symbol that from now on we will use to represent the Inhibit (I) operator. Panel (ii) shows the implementation of Inhibit in a purely digital context. The waveform in Panel (iii) depicts Inhibit's functionality through two examples: b Inhibits a and a Inhibits b.

As for the Inhibit function, inspiration was drawn from the behaviour of inhibitory post-synaptic potentials in the neurons of the neocortex [15]. More specifically, Inhibit works as a non-linear filter that has two inputs: an inhibiting signal and a data signal (that gets inhibited). If the inhibiting signal arrives first or at the same time as the data signal, the output is prevented from ever going high (no state transition), which is understood as a temporal ∞. However, if the data signal arrives before the inhibiting signal, the former is allowed to pass through the gate without any interruption. Figure 2.3 shows (i) the symbol used for b inhibiting a, (ii) its implementation with an augmented S-R latch, and (iii) a waveform depicting its functionality through two examples. An even more efficient implementation, consisting of a pass gate or a single PMOS pass gate, is also possible with a little customization [5].

With this set of temporal operators and codes, new trade-off and optimization spaces open up. Particularly, the very low number of wires and bit flips required by race logic circuitry promise very high energy efficiency. Few wires are required because each of them holds a multi-valued signal, the delay. Regarding switching activity, those wires flip from 0 to 1 at most once through a logic evaluation as the signal-front washes across the circuit. While not all computations map well to such an encoding, those that are have the potential to operate with very little energy. An open question answered in this chapter is if such a logic is applicable to any general learning or classification task.

2.3 Race Trees

While monolithic neural networks receive the lion's share of attention from the architecture community, with respect to machine learning, decision trees have proven to be incredibly useful in many contexts and a promising solution towards explainable, high-performing AI systems. A decision tree creates a hierarchy of decisions, which consists of a set of leaves (labels) and a set of decisions (branches). One normally starts from the top, where the tree's root node is found, and branches down the tree until a leaf is reached—see Figure 2.4 (i).

2.3.1 Reverse Race Trees

Existing race logic implementations, such as the genome sequence accelerator [14], perform computation by observing the relative propagation times of signals injected into the circuit. Following this example, one approach to implement decision trees is by virtually turning them upside down; we can think of them as reverse tree networks that route possible packets from the leaves to the root. Initially, a unique delay-encoded label is assigned to each leaf. These labels then race against one another and, where two of them meet, only one is allowed to propagate further. In the end, only the label associated with the correct leaf survives—the packet at the output of the network is unchanged, while all others get discarded along the way.

A decision tree that we use as an example case is presented in Figure 2.4 (i). Figure 2.4 (ii) shows the flow of the four delay-coded labels in the reverse tree for $x = 2$ and $y = 3$, with the corresponding waveform being illustrated in Figure 2.4 (iii). Its race logic implementation is depicted in Figure 2.4 (iv). The upper two blocks, colored in red and blue, correspond to the tree's internal nodes, $n1$ and $n2$, and are implemented with the use of one Inhibit and one FA operators. The bottom one, colored in yellow, is slightly more complicated as the label coming from its $False$ path can take more than one values—either label C or label D.

Note that when reversing a tree, the *if* clauses in its nodes must be revised. For instance, the threshold function in the $n2$ node, $y < 2$, must be rewritten as $y + 1 < 3$ for label $D = 2$. To implement $y + 1 < 3$, variable y must be delayed by one clock cycle. As discussed above, in a synchronous setting, a shift register can be used to perform constant addition.

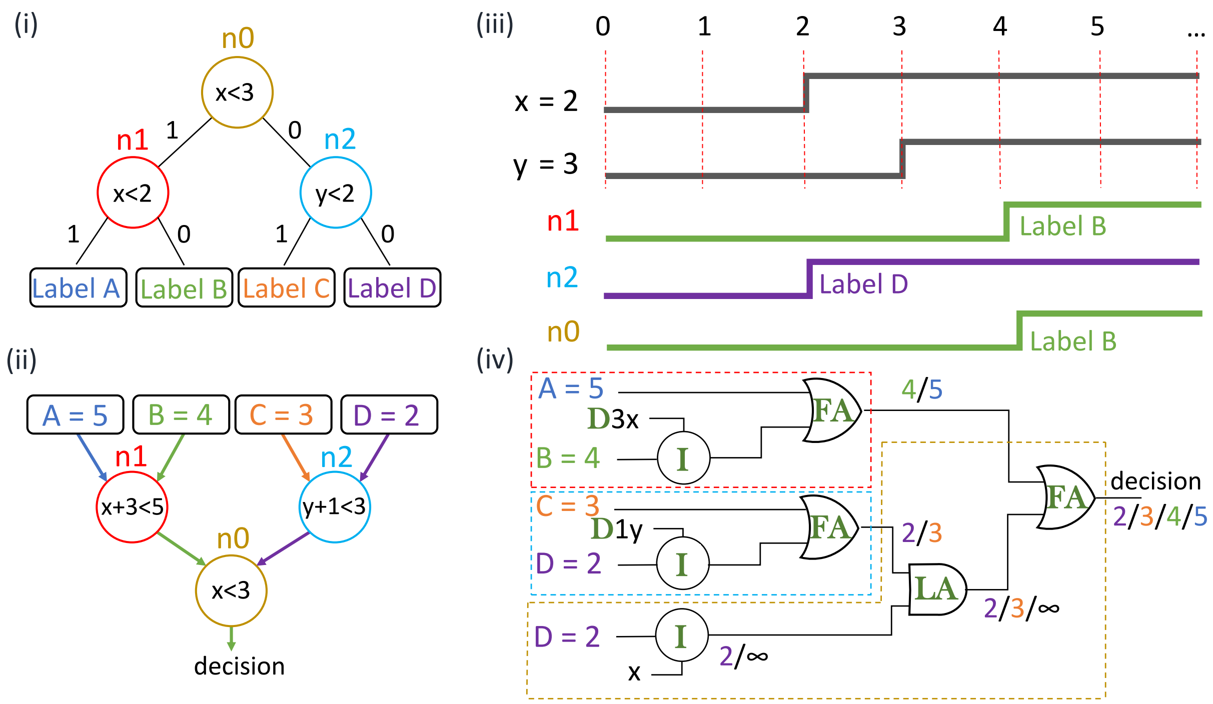


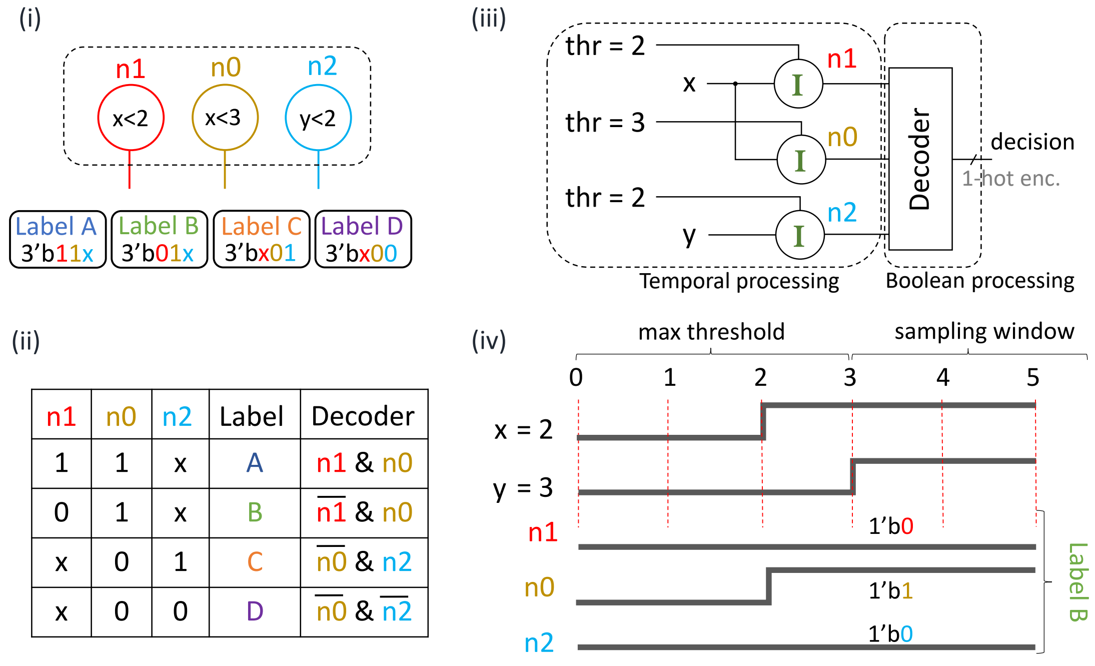
Figure 2.4: Panel (i) shows an example decision tree. Panel (ii) presents its "reverse" equivalent, as well as the flow of the four delay-coded labels for $x = 2$ and $y = 3$. Panel (iii) displays the corresponding waveform. Panel (iv) depicts the race logic implementation of this reverse tree. The leaf label associated with the *False* branch of a node plays the role of a in the Inhibit operator of Figure 2.3, whereas the record's attribute (x or y in this example) serves as the inhibiting input b. Given that subtraction and variable addition are not natively supported by race logic, the attribute routed to an Inhibit's controlling input should be adjusted accordingly; e.g., $y < 2$ is rewritten as $y + 1 < 3$.

2.3.2 Flat Race Trees

An alternative way to look at a decision tree is as a set of independent and parallel, rather than sequential, decision rules that lead to a final prediction [17]. Each leaf now can be represented as a logical function of the binary decisions encountered at the nodes on its path to the root. In other words, the tree gets "flattened" and each path from the tree root to a leaf corresponds to a unique combination of attribute test outcomes. The big idea behind the parallel execution of all these independent *if* clauses is shown in Figure 2.5 (i). For example, the leftmost leaf is reached only when both $n0$ and $n1$ return

$True$, while the output of $n2$ is inconsequential. The order that the outcomes of these conditions appear to reveal does not affect the final decision. Figure 2.5 (ii) presents the truth table describing the functionality of the decoder that associates node decisions with one of the leaf labels.

Figure 2.5 (iii) presents the implementation of such a flat tree in race logic. In contrast to conventional digital logic approaches, where the size of the circuit realizing the desired threshold functions (each node is a binary decision) are directly related to the resolution of the associated attribute and threshold values, this is not the case here. Each node is implemented with a single Inhibit gate.



n1	n0	n2	Label	Decoder
1	1	x	A	n1 & n0
0	1	x	B	$\overline{\text{n1}}$ & n0
x	0	1	C	$\overline{\text{n0}}$ & n2
x	0	0	D	$\overline{\text{n0}}$ & $\overline{\text{n2}}$

Figure 2.5: A decision tree can be viewed as a set of independent decision rules that lead to one and only one leaf when combined accordingly. These functions can be executed in parallel, as shown in Panel (i). Panel (ii) presents the truth table that describes how node decisions associate with leaf labels. Panel (iii) depicts the race logic implementation of this flattened decision tree with Inhibit gates, where thresholds serve as controlling inputs. Panel (iv) displays the resulting waveform for $x = 2$ and $y = 3$.

Moreover, because the decisions related to the various tree paths are mutually exclusive and the maximum threshold value is statically known, the transition from the temporal domain to binary happens seamlessly, without the need for any special circuitry. Figure 2.5 (iv) shows the resulting waveform for $x = 2$ and $y = 3$. In the given example, the maximum threshold value is 3; thus, all node decisions can be safely considered final after 3 cycles. We can safely sample the outcome of $n0$, $n1$, and $n2$ conditions at any time after that.

2.4 End-to-End Architecture

2.4.1 From Sensor to Delay Coded Input

Whenever a different encoding is considered, the cost of getting in and out of this new domain has to be taken into account. In the prior section, we discussed the output part, which includes the temporal-to-binary conversion and does not require any additional hardware resources. Here, we focus on the input part and show that race logic is a natural fit for in-sensor computing. Figure 2.6 presents an end-to-end architecture for temporal processing.

Because sensory input is analog in nature, most sensors begin with a measured voltage or current, which is then converted to a digital output with the use of ADCs. ADCs traditionally return digital binary values, necessary for Boolean processing. However, in race logic, processing happens in the time domain and information is encoded in relative delays instead of $True$ or $False$ values. This difference simplifies the design of sensor-processor interface components significantly. For instance, the costly time-to-digital conversion (TDC) in the ADCs is, in this case, redundant and can be skipped [18]. Moreover, further gains are expected for sensing systems that provide directly delay-coded outputs, such as dynamic vision sensors (DVS) [19], asynchronous time-based image sensors (ATIS) [20], time-to-first-spike (TTFS) [21] and time-of-flight (ToF) [22] cameras, and AER (address event representation) ear sound sensors [23]. On these occasions, the time-to-distance conversion can be entirely skipped.

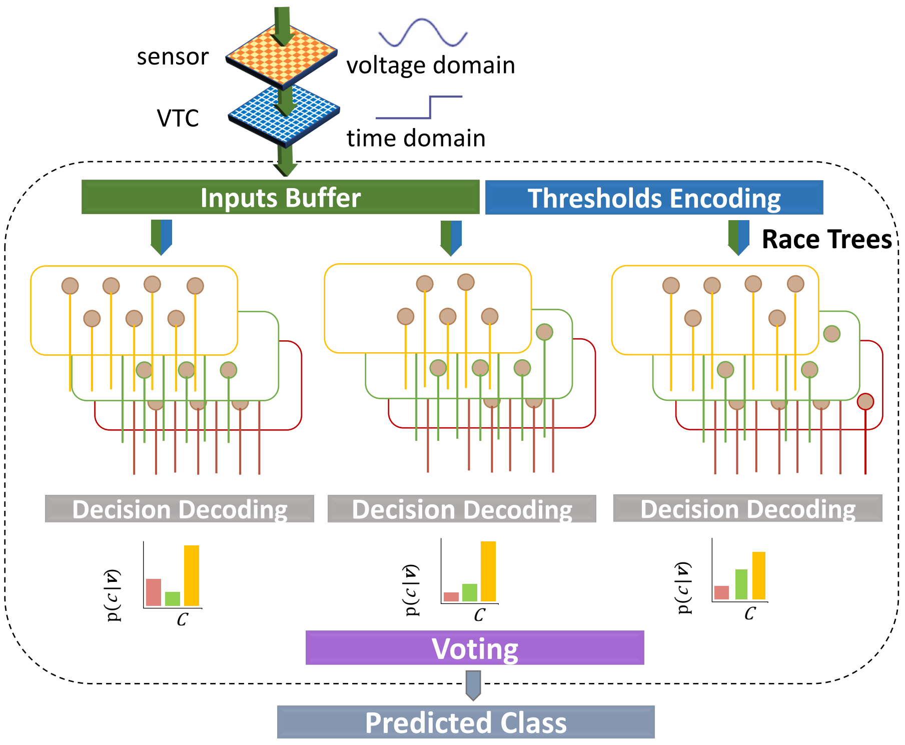


Figure 2.6: End-to-end temporal architecture for in-sensor processing.

2.4.2 Programmable Race Trees Architecture

Reverse and flat tree approaches show two ways of implementing decision trees in race logic. The reverse tree idea is of particular interest as it is unlike any other network. Typically, in a network, the packet contents are inert with respect to routing. For example, in the case of sorting networks, the packet values are used for routing, but they also have numerical content external to the network [24]. In reverse trees, the packets are externally assigned values that are symbolic and contain no useful numerical content—in much the same way that numbers in sudoku are used symbolically, but not numerically.

However, internal to the network, as part of the routing architecture, packet values do take part in numerical operations.

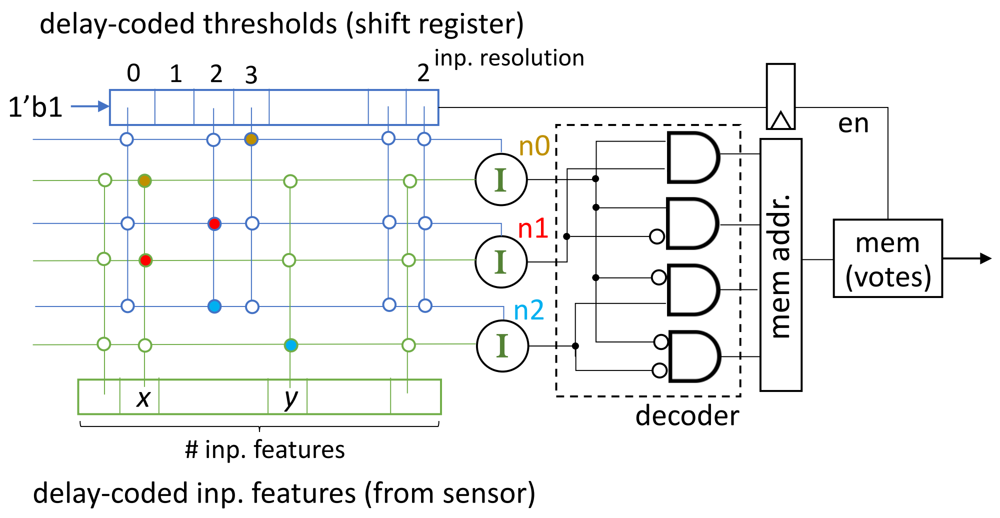

delay-coded thresholds (shift register)
0
1
2
3
2 inp. resolution
1'b1
n0
n1
n2
I
en
mem addr.
mem (votes)
decoder
x
y
# inp. features
delay-coded inp. features (from sensor)

Figure 2.7: Programmable race logic accelerator for a decision tree of depth 2. The length of the shift register, used for the threshold delay encoding, is defined by the resolution of input features. The memory block shown after the decoder is added to support the implementation of tree ensembles, where a weighted voting scheme follows.

The idea behind the flat tree approach is much simpler. This simplicity leads to a more compact and efficient hardware design, with fewer shift registers (necessary to perform constant additions) and a smaller interconnection network. Due to these reasons, we consider flat trees as our design of choice for the rest of this chapter.

Figure 2.7 presents a programmable architecture for the design of Figure 2.5. To ensure that it is possible to route any delay-coded threshold value and any input feature to the desired tree node, two programmable crossbar arrays are used. The decoder following the rank of Inhibits implements the truth table of Figure 2.5 (ii). Note that although Inhibits conceptually operate in the temporal domain, their sampled output is a typical binary vector. Thus, only one memory read is required per input record at the time that the propagating wavefront reaches the end of the shift register. Prior to the next computation, the circuit must be reset.

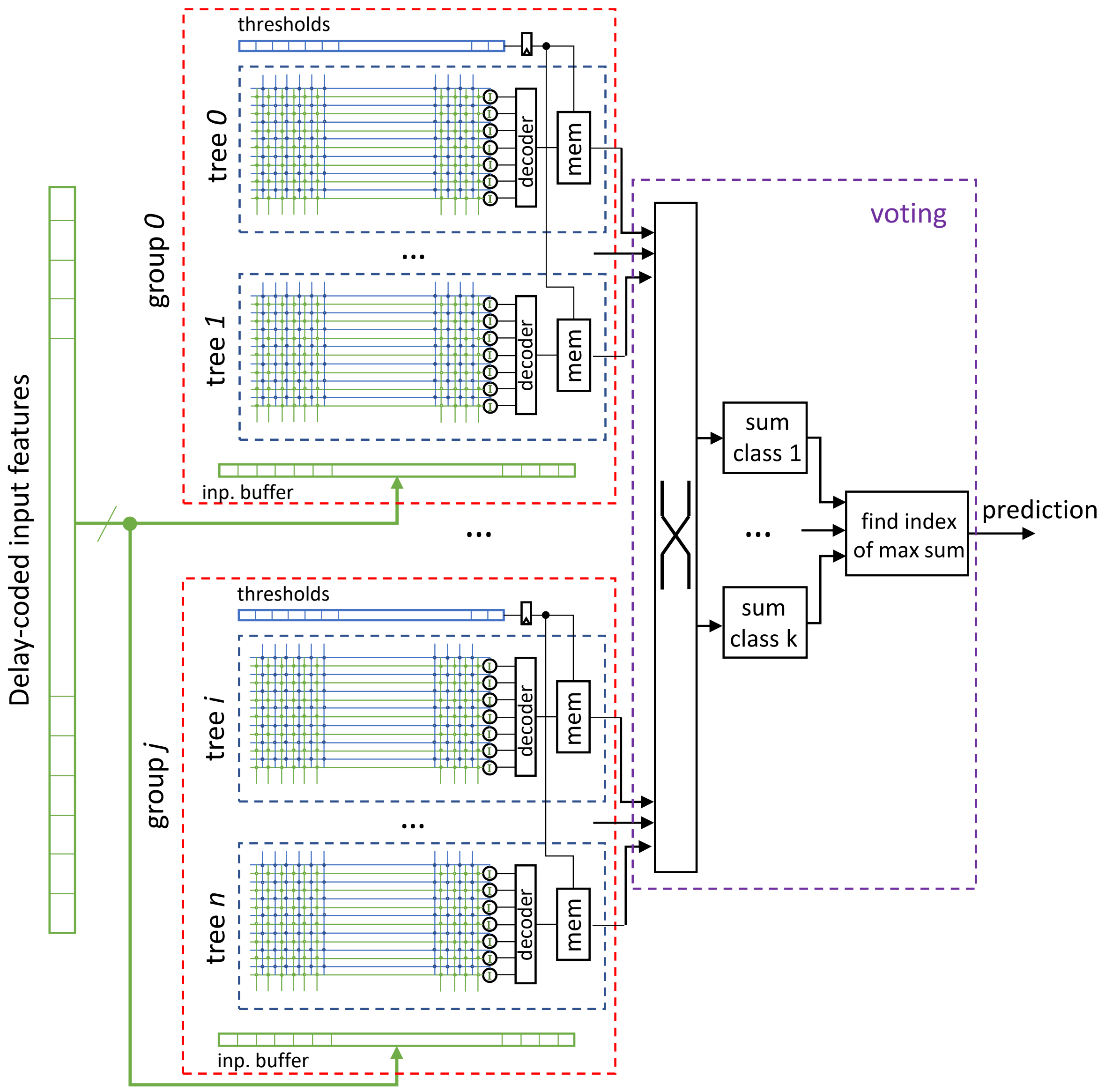


Figure 2.8: Programmable temporal accelerator for decision tree ensembles.

Figure 2.8 presents a high-level diagram of a programmable temporal accelerator for decision tree ensembles. To keep the overhead of the clocked components low, trees are organized into groups to minimize the number of shift registers and buffers needed for the generation of delay-coded thresholds and safe clock domain crossing (between the sensor and accelerator domains). Finding the optimal number of trees per group requires a trade-off analysis between the costs of clocked components and routing networks: adding more trees per group reduces the number of clocked components but increases the size of the crossbar arrays. For further efficiency, crossbars may also be replaced by more efficient configurable routing networks, without any effect in our system's functionality.

Finally, because the votes retrieved from memory are in regular binary encoding, the implementation of the weighted voting scheme is based on typical binary adders. Once all prediction values of all trees have been summed, a comparison between them takes place to find the class with the highest value and determine our system's final "guess".

2.5 Software Infrastructure

When designing a new system, development time and usability are major concerns. Ideally, the proposed architecture should be modular for systematic implementation, and an abstraction layer, hiding the hardware complexity from domain experts, should also exist. In the case of machine learning, the vast majority of development happens with the use of specialized software libraries. Here, we use the open-source scikit-learn framework [7] as a starting point and develop a fully automated toolchain that support hardware generation, verification, and analysis.

The user is expected to train an ensemble of desision trees in scikit-learn, as usual. Once the this tasks completes, the trained model go through a preprocessing stage where first the effects of data and vote quantization are analyzed and second the optimum number of trees per group (see Figure 2.8) is found. As a next step, the trained model, along with the generated metadata, are sent to a hardware generation engine. More specifically, in this stage, decision trees are parsed from top to bottom and dedicated hardware is generated for each node with the help of a library of temporal primitives. For the generation of the glue logic between them, hardware templates are used. The resulting description comes in Verilog and PyRTL [25] formats [1]. PyRTL is a Python embedded hardware design language that returns synthesizable hardware and it is particularly useful here because it allows for cross-checking validation against scikit-learn's *predict* function without leaving the Python environment. To simulate temporal input stimuli, Python's generators are used.

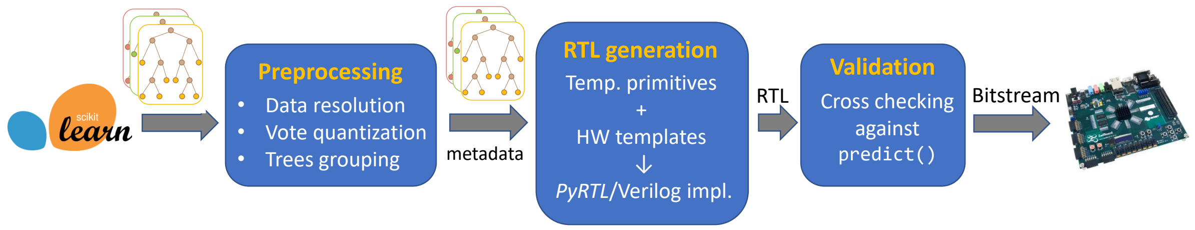


Figure 2.9: Overview of the developed software infrastructure. For the training part, the open-source scikit-learn framework [7] is used. The tool (a) provides the user with the options to quantize input features and votes, (b) supports the automatic generation of model-specific race trees circuitry directly from scikit-learn structures, and (c) assists design evaluation through cross-checking with software models.

[1]PyRTL does not allow the direct use of S-R latches. So, for the PyRTL implementation of Inhibit, we use an alternative sequential design instead.

2.6 Evaluation

2.6.1 Methodology

To evaluate the proposed design and identify opportunities for further improvement, analytical and empirical power and area models for the basic components of our architecture are created. Towards this end, we synthesize the RTL of each sub-component of race trees individually, as well as the RTL of the whole design, as that generated by the above-described software infrastructure. We rely on open-source tools [26, 27] and a publicly available 14 nm standard cell library [28] to obtain the desired area, power, and performance results. In the energy and throughput calculations that follow, we use a 500MHz clock and do not scale the attained power results to compensate for the lack of a wire load model. Under this assumption, each operation consumes twice its nominal energy [29]. Regarding crossbars, power numbers were retrieved from existing literature [30, 31].

2.6.2 Implementation Results

According to existing literature, MNIST is the most commonly used dataset in the context of proof-of-concept prototypes. The following analysis is based on these data. Note that the dataset selection primarily affects the prediction accuracy of the system, with negligible effects on the power consumption and speed of the system (as far as the model fits in the system configuration). In all cases, latency, in terms of clock cycles, is calculated as follows:

$$latency = 2^{inp\_res} + log_2(\#ests) + \#classes \tag{2.1}$$

where the first term is related to the maximum threshold value, whereas the second

Table 2.1: Synthesis results for hardwired race trees produced by Yosys [26] using a publicly available 14 nm cell library [28].

# Trees	**Depth**	**Inp. res.** (bits)	**Vote res.** (bits)	**Accuracy**	**Latency** (CCs)	**Power** (mW)	**Area** (mm^2)	**Freq.** (MHz)
1,000	6	8	8	97.48%	273	521	0.46	1,000
1,000	6	4	4	97.45%	33	475	0.45	1,000
200	8	4	4	96.18%	31	384	0.33	1,000
200	6	4	4	95.72%	31	125	0.13	1,000

and third terms are associated with the voting part. This means that, under the above assumptions, race trees can classify up to $1.83M$ images/s when 8 bit inputs are used and up to $16.1M$ images/s for 4 bit inputs.

Detailed results are provided in Table 2.1. According to these data, a classifier that consists of $1,000$ race trees of depth 6 attains 97.45% accuracy, while still maintaining a low energy expenditure at 31.35 $nJ/pred$. A more efficient approach, which utilizes only a fifth of the number of trees, achieves a performance of 95.7% with energy numbers as low as 7.8 $nJ/pred$. By increasing the depth of trees to 8, the accuracy increments by 0.5% at the expense of 16.1 nJ of additional energy per prediction. Note that for the training of the above race trees, Gradient Boosting [32], whose derivatives have recently gained popularity by winning various Kaggle and other data science competitions, was used.

2.6.3 Comparison with state of the art

In recent years, an explosion of hardware accelerated machine learning activity has resulted in a wide variety of ASIC architectures for comparison. Figure 2.10 plots a few of these solutions on an accuracy versus energy plot. All results are scaled to 28 nm.

One approach that has tried to tame the massive neural network accelerator design space is Minerva [35], represented by datapoint *c*. Minerva is an automated co-design approach that accounts for algorithmic, architectural, and circuit level constraints in

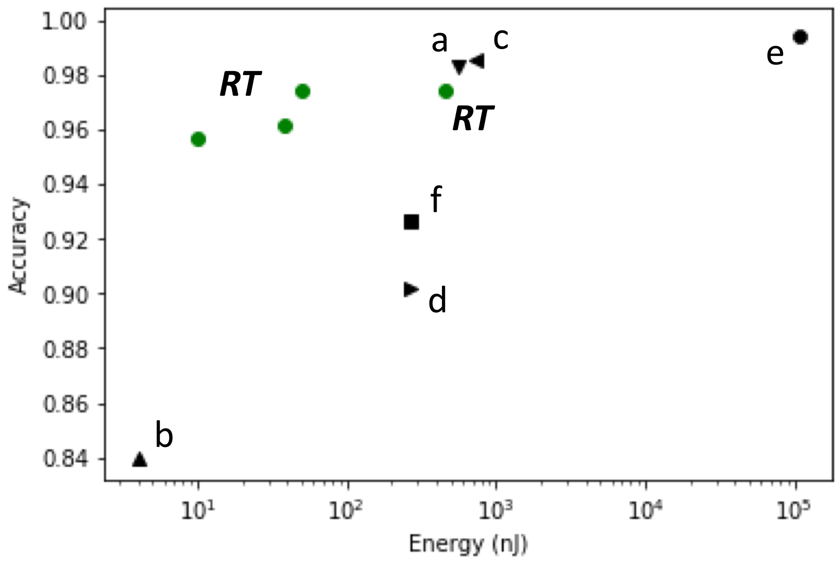


Figure 2.10: Accuracy vs energy comparison: a [33], b [34], c [35], d [36], e [37], f [37]. All results are scaled to 28 nm. Green dots represent race trees.

an attempt to efficiently accelerate deep neural networks. More specifically, Minerva first performs design space exploration at the algorithmic and architectural level and then tweaks the resolution and prunes certain unnecessary energy hungry operations. Moreover, it looks at circuit-level optimizations, such as SRAM fault mitigation, before reporting accurate chip level performance metrics. This broad design space exploration and multi-level optimizations allow Minerva to be highly accurate, still energy efficient, and make it a good starting point for comparison.

Another interesting approach is the sparse event-driven neuromorphic object recognition processor represented by datapoint b [34]. The design consists of a locally competitive algorithm (LCA) inference module [38], used for feature extraction, and a task-driven classifier. Its spiking neuron architecture allows for a very low energy cost, 20.7 $nJ/pred$ in 65 nm. On the downside, this implementation achieves only 84% accuracy, which is the lowest among the displayed solutions.

At the other extreme, a high-performance sparsely connected neural network running

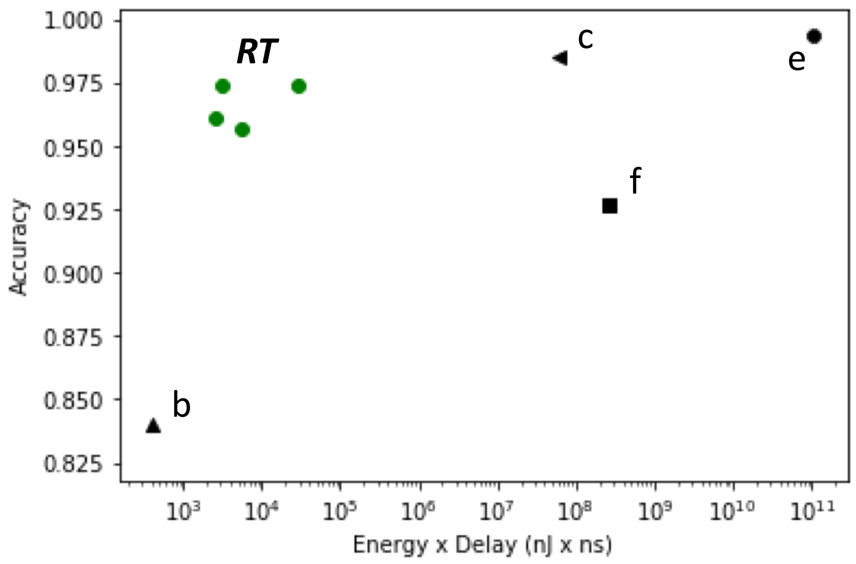


Figure 2.11: Accuracy vs energy-delay product comparison: b [34], c [35], e [37], f [37]. All results are scaled to 28 nm. Green dots represent race trees.

on the 28 nm IBM TrueNorth chip utilizes 64 ensembles and hits very high accuracy numbers, 99.42%, at the the expense of 108 $\mu J/pred$ [37]. A more energy-efficient version with a single ensemble is also reported and achieves 92.7% accuracy at 268 $nJ/pred$. These implementations are represented in Figure 2.10 by datapoints *e* and *f*, respectively. A few other solutions with comparable accuracy and energy performance, represented by datapoints *a* [33] and *d* [36], are displayed, too.

Our race trees are represented in green dots. As can be seen, the configuration that consists of 1,000 trees of depth 6 and uses 8 bit data achieves comparable energy efficiency and accuracy to datapoints *a* and *c*. If 4 bit data are used instead, the accuracy remains almost unchanged, but the energy efficiency of race trees improves by more than 10x. Further gains are possible with smaller race trees designs. In all cases, race trees capture the top left corner of the graph and are Pareto optimal.

To get a more holistic view of the landscape, Figure 2.11 also provides a comparison in terms of energy-delay product. This analysis reveals that the efficiency gap between race trees and their counterparts is actually even bigger. In other words, race trees do not only improve energy per operation but also achieve lower latency due to their increased parallelism, short critical paths, and feedforward model of execution without hardware time-sharing.

2.6.4 Component breakdown analysis

Our last evaluation goal is to understand where the majority of power and area goes in race trees. Tables 2.2 and 2.3 provide estimates for each of race trees' main blocks. As a sanity check, we compare the sum of area and power results of all these components for each classifier against the complete synthesized design results, shown in Table 2.1. The differences observed are expected as all the trees are now considered fully-grown and programmable [30, 31] to cover the most general case.

Table 2.2: Estimated power consumption for various sub-units of the race trees architecture.

# Trees	**Depth**	**Inp. res.** (bits)	**Vote res.** (bits)	**# Trees per group**	**Tech node** (nm)	**Thresholds** (μW)	**Inp. Buffers** (μW)	**Trees & Dec.** (μW)	**Memory** (μW)	**Voting** (μW)	**Progr. Intercon.** (pW)	**Total** (mW)
1,000	6	8	8	100	14	7,250	22,000	529,600	111,500	2,000	0.252	673
1,000	6	4	4	100	14	360	22,000	529,600	59,300	1,700	0.016	613
200	8	4	4	20	14	360	22,000	452,700	35,750	320	0.013	511
200	6	4	4	20	14	360	22,000	105,900	11,900	320	0.003	140

Table 2.3: Estimated area for various sub-units of the race trees architecture.

# Trees	**Depth**	**Inp. res.** (bits)	**Vote res.** (bits)	**# Trees per group**	**Tech node** (nm)	**Thresholds** (mm^2)	**Inp. Buffers** (mm^2)	**Trees & Dec.** (mm^2)	**Memory** (mm^2)	**Voting** (mm^2)	**Progr. Intercon.** (mm^2)	**Total** (mm^2)
1,000	6	8	8	100	14	1.0e-2	3.5e-2	0.5	0.05	0.03	-	0.65
1,000	6	4	4	100	14	7.5e-4	3.5e-2	0.5	0.03	0.03	-	0.60
200	8	4	4	20	14	7.5e-4	3.5e-2	0.4	1.5e-2	6e-3	-	0.45
200	6	4	4	20	14	7.5e-4	3.5e-2	0.1	5.5e-3	6e-3	-	0.15

2.7 Conclusion

As machine learning techniques continue to find new and compelling applications across a wide range of computing tasks, the desire to bring them into even our lowest-power devices will only continue to grow. Applying these complex algorithms without resorting to the use of significant amounts of energy remains a challenge.

In this chapter, we focused on in-sensor classification and showed the natural relationship between decision tree algorithms, race logic, and the underlying sensors themselves. Others have already studied the analog advantage of avoiding the final step of converting input signals to a pure digital representation [39]—instead leaving them as a variable delay that can be trivially converted to a "race" encoding. At the algorithm level, little is needed in the way of changes other than reimagining the configuration of existing decision tree models. At the architecture level, the improvements are dramatic both in hardwired and programmable configurations. The resulting race trees design has a shallow critical path, supports a feedforward model of execution with dedicated hardware resources, and prompts exceedingly few bit transitions as computation propagates through the circuit.

While the resulting system already performs admirably well with regards to energy, area, and performance, there is still room for further exploration and improvement. The evaluation here has yet to take advantage of (a) the asymmetric nature of the logic-level transitions, (b) the malleability afforded by machine learning algorithms, and (c) the opportunity to implement Delay and Inhibit functions even more efficiently through customization. Lastly, the integration of race logic accelerators with other circuits operating purely on the time-domain [40] is another interesting path for exploration towards the construction of more complicated systems.

Chapter 3

Temporal Superconductor Computing

3.1 Introduction

Advancements in semiconductor electronics have driven improvements in computer performance and efficiency for more than 50 years. However, as scaling becomes increasingly challenging and the resultant benefits diminish, questions about ways to leverage new devices arise. Superconductor electronics provide an appealing alternative as they enable systems with zero static power dissipation, speed-of-light energy-efficient interconnects, and clock speeds in the 10s or 100s of GHz. Even more importantly, superconductor electronics can also serve as facilitators for integrated classical-quantum computers due to their cryogenic nature and above characteristics.

Despite their theoretical promise, the development of meaningful superconductor computer systems in practice has proven to be tricky. A semantic gap lays between level-driven logic, which semiconductor designs accept as a foundation, and the pulse-driven logic naturally supported by the most compelling superconductor technologies.

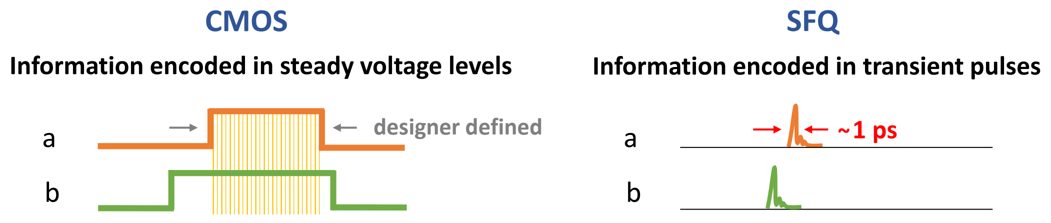

CMOS
Information encoded in steady voltage levels
a
b
designer defined
SFQ
Information encoded in transient pulses
a
b
~1 ps

Figure 3.1: In CMOS, information is typically encoded in steady voltage levels, the duration of which is a controlled variable. In single flux quantum (SFQ) technology, latching in a similar way is not possible, and thus information is encoded in transient pulses.

This gap creates a variety of challenges across the full hardware stack, from the circuits up to the tools and architecture levels.

A pulse, unlike a stable voltage level, will fire through a channel for only an instant (see Figure 3.1). Arranging the network of superconductor components so that input pulses—driven by the transfer of magnetic flux quanta—always arrive simultaneously to logic gates is obviously not a possibility. An approach commonly used in single flux quantum (SFQ) logic systems is to consider the presence of a pulse during a given time interval as a logical 1 and the lack of a pulse as a logical 0. This convention requires two things of SFQ circuits: first, all logic gates must agree on a prescribed time interval for evaluation; second, they must all be able to remember whether or not a pulse has arrived during the interval. The second requirement fits nicely with the inherently stateful nature of superconductor cells composed of superconducting quantum interference devices (SQUIDs, or loops formed by two Josephson junctions (JJs) and one inductor), which hold SFQ pulses. However, the first requirement is more difficult to fulfill and is traditionally met through extremely fine-grained clocking (see Figure 3.2).

A deeper look into this fully synchronous reality unveils that maintaining the illusion of a strictly Boolean evaluation is not only a significant engineering hurdle but also results in unavoidable overheads. We claim that many of these issues can be resolved by leaving

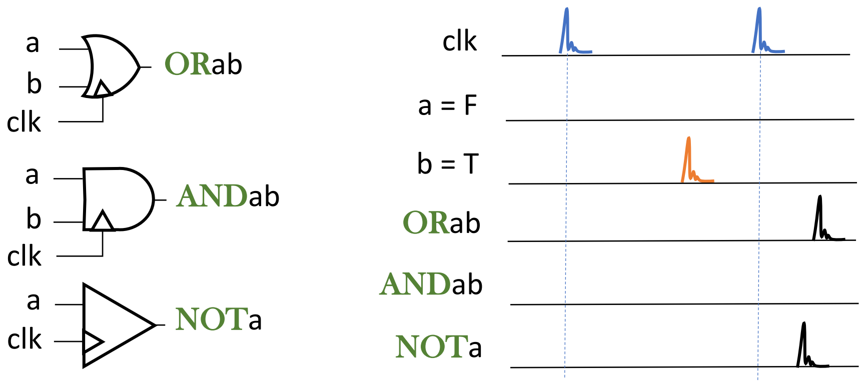


Figure 3.2: Boolean (clocked) SFQ cells. If a data pulse appears in a *clk* interval is understood as a logical 1; otherwise, it is a logical 0. The arrival of a *clk* signal releases the content of the cell and resets it.

the premises of Boolean logic and adopting a new way of looking at those pulses. More specifically, if we instead think about pulses as the natural representation of data, the natural language for expressing computations over that data would be one that could efficiently describe the temporal relationships between their arrivals.

To explore this idea, we draw upon the work presented in Chapter 2, decouple race logic from prior technology-dependent assumptions, and show how its principles can be directly applied to problems in superconducting. This approach allows to remove clock from superconductor gate semantics, design a library of clock-free temporal SFQ (single flux quantum) cells, and create useful new architectures. To the best of our knowledge, this is the first time that race/temporal logic is used to specify computations with superconductor electronics.

3.2 Computing with Superconductors

3.2.1 Fundamental concepts

Superconductor electronics are defined by three basic features: (a) the absence of resistance in static circuits at superconducting temperatures, (b) the Josephson effect, which governs the fundamental switching element in superconductor circuits, the JJ, and (c) the propagation of single flux quanta[1], instead of static voltage levels as in CMOS. As a two-terminal device, the JJ does not switch in the same way as three-terminal CMOS transistors. Normally, current flows through the JJ with no impediment, like a zero-resistance wire. However, at a threshold called the critical current, the resistively-shunted JJ blocks off the current flow for a short time as it switches, thereby creating an SFQ pulse on the JJ output. Each pulse is a short burst of magnetic energy observed through a change in voltage.

3.2.2 Opportunities

One of the most compelling arguments for the use of superconductor electronics is their potential to significantly surpass end-of-roadmap CMOS circuits based on energy-delay product, even when the overhead due to cooling is considered [41]. SFQ circuits achieve $10-100$ times higher clock frequencies than CMOS, and the switching energy of an individual JJ is $\sim 10^{-19}$ J. Energy-efficient versions of the SFQ technology, such as ERSFQ [42], eliminate static power dissipation without sacrificing speed or circuit-level equivalence to the more traditional RSFQ (rapid SFQ) [43]. A recently-proposed AC/SFQ powering scheme also reduces bias requirements by locally storing small currents from rectified AC voltage to power SFQ gates [44]. On top of that, transmission of SFQ pulses across superconducting wires requires very little energy [45] because no RC charge process is involved. All these facts, along with the stateful nature of superconductor elementary cells, give designers the opportunity to explore a fundamentally different trade-off space, reevaluate existing architectural solutions, and exploit the unique characteristics of superconductor electronics for the development of innovative computing

[1]One magnetic flux quantum is 2.07×10^{-15} Wb or $2.07\,\text{mV} \times \text{ps}$ in units more familiar to computer architects.

machines.

Besides speeding up and reducing the power consumption of CMOS circuits for classical applications, superconductor electronics open pathways for scaling up quantum computers. More specifically, the ability of superconductor circuits to operate at very high speed enables the fast processing of qubit output data for error correction and qubit control [46]. Their low power overhead and cryogenic operating temperature also allow them to reside next to the quantum processor, thus eliminating the control cables that leave the cryogenic environment and introduce thermal noise.

3.2.3 Challenges

Despite their advantages, superconductor electronics pose a number of challenges. One of the most profound originates from the pulsed-based nature of computation. Pulses cannot be sampled like voltage levels because they do not coincide with picosecond precision. Thus, methods developed for transistors and other latching circuits do not carry over easily.

Moreover, an SFQ pulse is fundamentally discrete. Because of this quantization, a fan-out that produces two pulses from a single pulse requires an active component called a splitter. As a consequence, signals with significant fan-out inflate circuit size considerably. To make matters worse, the relatively high process variability in superconductor electronics [47] can skew signals significantly across large fan-out trees, leading to synchronization problems, additional logic overheads, and reduced operating speeds.

Finally, the lack of a reliable and high-capacity memory operating at 4.2 degrees Kelvin, the default temperature for superconductor electronics, imposes its own distinct limitations. Recent studies indicate that cold memories built from CMOS DRAM operating at 77 degrees Kelvin offer a promising solution in the near future [48], while advancements in fabrication processes are encouraging for competitive superconductor memories in the longer term [49]. In both cases, the expected gap between the access latency of memory and the high operational speed of SFQ logic circuits introduces challenging microarchitectural problems.

Because of these unique characteristics (and limitations) of superconductor electronics, research on computing paradigms and architectures that depart from classic CMOS-inspired solutions, and can potentially (a) use much fewer JJs than transistors for the same information throughput, (b) allow for easier clocking, and (c) have lower memory requirements, offers a promising direction for innovation [50]. We claim that temporal computing opens up precisely this research space.

3.3 Temporal Operators in SFQ

The way in which events are encoded plays a critical role in selecting the hardware that most efficiently implements logic operators. For example, as discussed in Chapter 2, single OR and AND gates can be used for the CMOS implementation of the FA and LA operators. Unfortunately, this hardware mapping does not make much sense in superconducting. More specifically, 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 a high state will not make a transition back to low in the same computation. However, 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.

A potential solution to this problem is to embed the state into each gate. Interestingly, the majority of SFQ elementary cells have both logic and storage abilities [43]; thus, they can be thought of as simple state machines. To facilitate the implementation of temporal

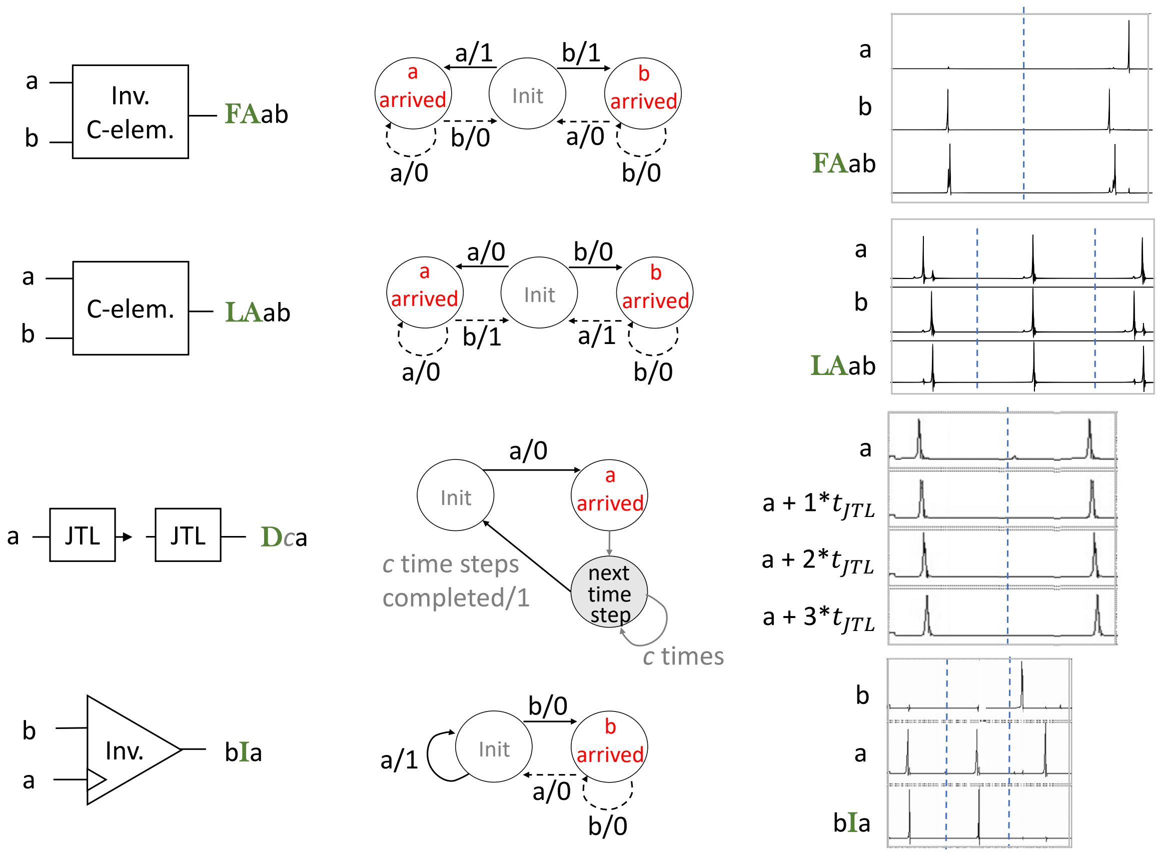


Figure 3.3: Block diagrams, Mealy machine representations, and SPICE-level simulations of FA, LA, D, and I temporal operators implemented in SFQ.

operators to stateful SFQ elements and show that no clock signal is needed, we first draw FA, LA, D, and I as Mealy machines. Figure 3.3 provides the corresponding illustrations. Then, we follow these Mealy machine descriptions and build FA by using an inverted C-element cell [43], LA using a C-element cell [43], D using a a sequence of Josephson transmission lines (JTLs) [2] and I using an Inverter cell. Area and latency results for each of these operators are provided in Table 3.1. The shown estimates are based on SPICE-level simulations using the MIT Lincoln Laboratory SFQ5ee process [47].

[2]The length of a JTL chain depends on the interval duration that associates with each time unit. Another possible implementation is with destructive readout (DRO) cells; however, this approach requires clocking.

Table 3.1: Area and latency estimates of temporal SFQ cells implemented in the MIT Lincoln Laboratory SFQ5ee process [47].

Element	Area (#JJs)	Latency (ps)
FA	6	12
LA	6	8
D	2/JTL	5/JTL
I	8	11

3.3.1 Resetting

Given the absence of an explicit clock/reset signal and the stateful nature of SFQ cells, resetting must also be rethought. For example, an SFQ Inverter, which implements Inhibit, will not return to its initial state until a pulse arrives to its clock port, while C- and inverted C-elements, used as Last Arrival and First Arrival gate, will not reset until both their inputs arrive.

Finding a universal and scalable resetting solution still remains an open challenge. We consider three possible cases, each of them coming with its own advantages and disadvantages: (a) thermal/power cycling—this is slow but requires no additional hardware resources; (b) emission of additional SFQ pulses in the existing hardware setup—this requires no additional resources and seems more practical than thermal/power cycling but applies only to LA and FA cells; (c) leaky gates [51]—this is applicable to all temporal cells but requires several challenges to be addressed to achieve slow leak rates.

Note that, in some cases, it may also be possible to replace stateful with stateless cells, therefore eliminating the need for resetting completely. Examples can be found in Section 3.5.

3.4 Circuit Design with Temporal SFQ Gates

Building from these SFQ implementations, the next step is to solidify the foundation for the design of complex superconductor temporal circuits. As discussed, clocking and synchronization are two of the most critical concerns and limitations in the design process of a SFQ circuits. Thus, here, we primarily focus on those.

As mentioned above, Boolean SFQ gates are sequential rather than combinational circuits. Hence, the designer must tightly synchronize each of them with all other gates and the clock network. Satisfying this requirement comes at a price; e.g., inflated circuit size (partially due to the expensive clock distribution network and the need to delay pad uneven paths), low tolerance to jitter and skews (particularly critical in high-speed designs implemented with processes that exhibit high variability), and increased energy consumption (a significant number of additional SFQ pulses is needed to carry a data signal across the circuit). Unlike Boolean circuits, this is not the case for temporal designs, as the clock is no longer part of the gate semantics.

The use of synchronous components may sometimes be beneficial, however. For instance, although a Coincidence function, which returns an output pulse if both a and b inputs arrive within the same interval, can be built from the above-described temporal operators [15], a more efficient implementation is possible: all that us needed for its superconducting realization is a synchronous AND gate. Even in cases where the use of synchronous SFQ blocks may be preferred, temporal designs allow the use of a data-driven self-timing (DDST) scheme, as shown in Figure 3.4.

In a DDST scheme, timing information is carried by data. More specifically, the required clock signal is locally generated by a logical OR function—implemented in the provided example with a merger SFQ cell—between the two input data lines. To extend the evaluation window of the synchronous logic block (defined by the delay between data

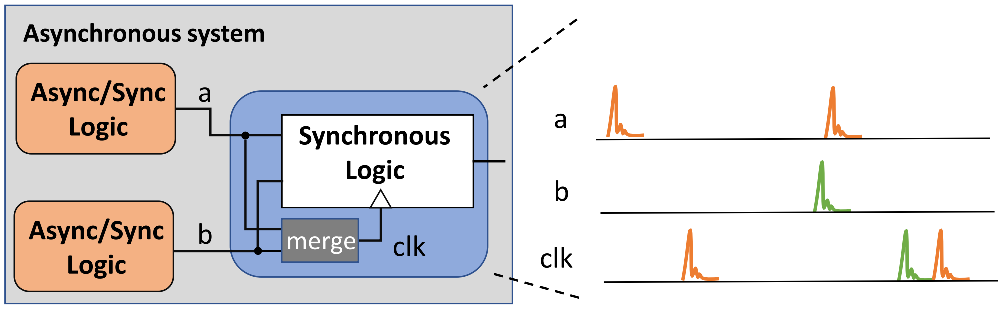


Figure 3.4: Proposed data-driven self-timing (DDST) scheme. The clock signal is generated from input data locally. If no input pulse arrives, it is safe to assume that the operator is idle, and thus no clock pulse is needed.

and clock pulse arrivals), JTLs can be added after the merger cell. Note that such a DDST scheme is not directly applicable to Boolean circuits as not all Boolean gates (e.g., NOT) are idle for the time steps where no input pulses arrive [3].

3.5 Evaluation

For the evaluation of the proposed logic scheme and methodology, we design, simulate, and measure the performance of various superconductor temporal accelerators. Detailed descriptions of race trees and other proof-of-concept accelerator designs follow.

3.5.1 Experimental Setup and Design Principles

We perform circuit simulations and analysis in both the open-source WRSPICE [54] and Cadence Spectre platforms using the MIT Lincoln Laboratory SFQ5ee process [47]. For gate isolation, path delay padding, and cell interconnection, JTLs are used. For fan-out, we use splitter cells. Finally, we define minimum Δt, which denotes the duration of each unit time interval, based on the principles of wave-pipelining.

[3]DDST schemes have been explored for Boolean circuits, as well [52, 53]. However, the need to cover NOT and its variants imposes the use of dual-rail codes, which, in this case, come with a significant overhead.

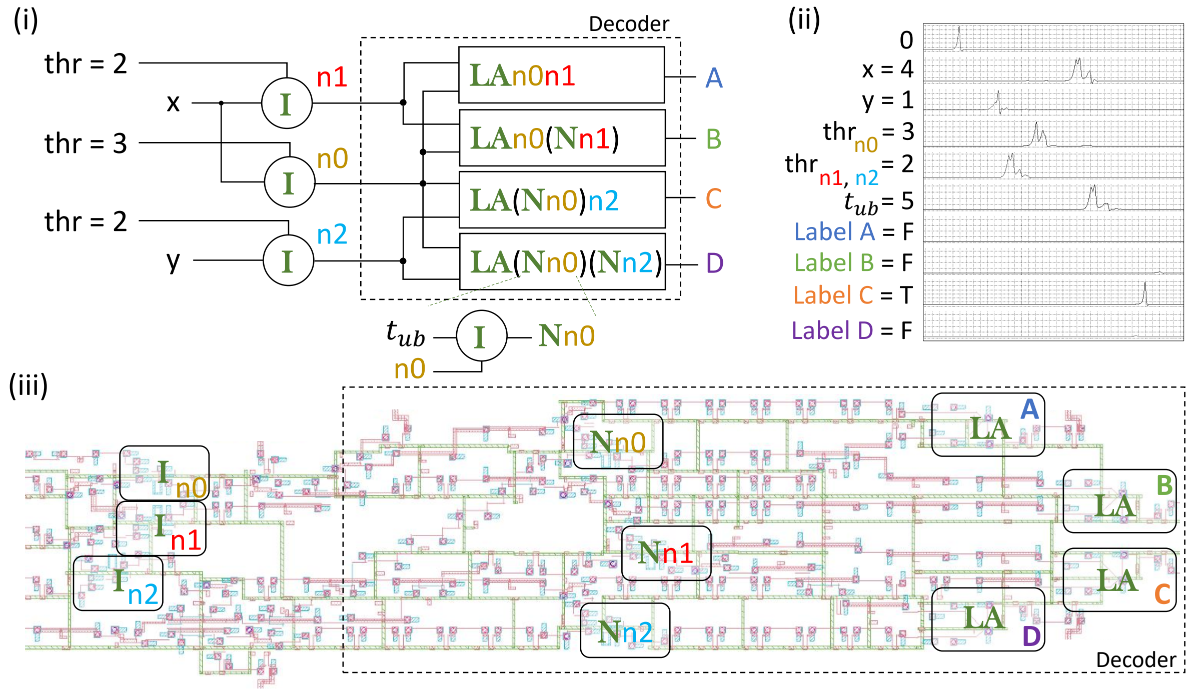


Figure 3.5: Panel (i): block diagram of a SFQ race tree. Panel (ii): Simulation results for $x = 4$ and $y = 1$. Panel (iii): layout diagram (unlabelled JJs are used for interconnection, splitting, routing, and testing purposes).

3.5.2 Proof-of-Concept Designs

Race Trees

In the case of race trees (see Chapter 2), each tree node is considered an independent temporal threshold function and realized with a single Inhibit operator. Figure 3.5 (i) shows the block diagram of a SFQ race tree corresponding to the model illustrated in Figure 2.4 (i).

For the construction of NOT gates, required for the implementation of the label decoder, Inhibit cells with an upper bound reference signal t_{ub} are used. This reference signal denotes the end of one round of computation (directly related to inputs resolution). Thus, NOT will fire an output pulse at time $t = t_{ub}$ if and only if it has not received any input pulses from time reference 0 until that moment.

Figures 3.5 (ii) and 3.5 (iii) provide simulation results and a layout diagram of this design. In the shown example, x and y are set to 4 and 1, respectively, and t_{ub} is equal to 5. As expected, the final outcome (1-hot encoded) is Label C. The total latency, for $\Delta t = 25$ ps, is 150 ps, and the design consists of 164 JJs: 72 JJs for logic elements, 24 JJs for splitters, and 68 JJs for JTLs. The number of JJs in the layout is greater than 164 because of the additional cost of routing and the inclusion of testing circuitry. More results and a comparison with CMOS are provided in Table 3.2.

Table 3.2: Estimated latency results for hardwired race trees in both CMOS (*f=1* GHz) [5] and SFQ ($\Delta t = 25$ ps).

Depth	Input res.	CMOS Latency	**SFQ Latency**	**Improvement**
6	4 bits	17 ns	0.464 ns	37×
6	8 bits	257 ns	6.464 ns	40×
8	4 bits	17 ns	0.490 ns	35×
8	8 bits	257 ns	6.490 ns	40×

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 the alignments with the highest or lowest score. To find these score, for two arbitrary strings P and Q, a 2D grid is constructed (see Figure 3.6 (i)). Each pair of letters, one from P and one from Q, associates with one of the following three operations: deletion, insertion, or match. Each of these operations is represented with a directed edge and may have its own score.

In the temporal version of the algorithm, each score corresponds a delay. Hence, the total time required for an input pulse to propagate from the top left to the bottom right corner of the grid reveals the obtained similarity score. The architecture of this accelerator can be generally thought of as a systolic array, where each cell is implemented in SFQ,

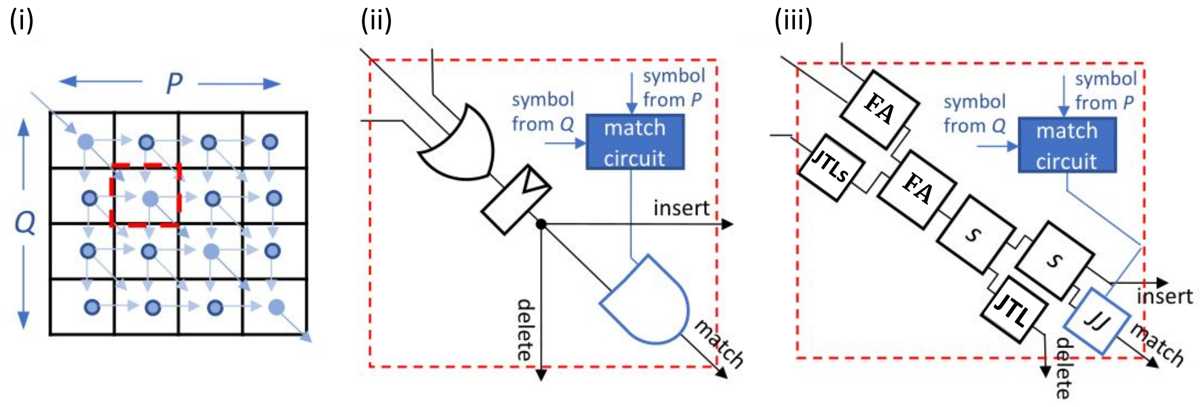


Figure 3.6: Panel (i): 2D grid constructed following Needleman and Wunsch's algorithm. Panel (ii): schematic of a CMOS unit cell for the temporal implementation of a genome sequencing accelerator. Panel (iii): SFQ equivalent circuit. P and Q represent the two strings to be aligned. The penalties/delays for deletion, insertion, and match are set to 1.

as shown in Figure 3.6 (iii). Figure 3.6 (ii) illustrates its CMOS implementation [14] as a reference.

To completely eliminate the need for clocking in the SFQ version, no synchronous components are used. The delay that corresponds to a score/weight of 1 comes from the propagation delay of each unit cell. To balance the delays between the various paths within each unit cell, we rely on Josephson transmission lines (JTLs). To control the propagation of a pulse across the diagonal, which should only happen when a match occurs, a JTL's bias is either turned on or off. For example, if there is a mismatch, the bias is low and an incoming SFQ pulse cannot cause the JJ to fire.

Figure 3.7 shows WRSPICE simulation results for a 3×3 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

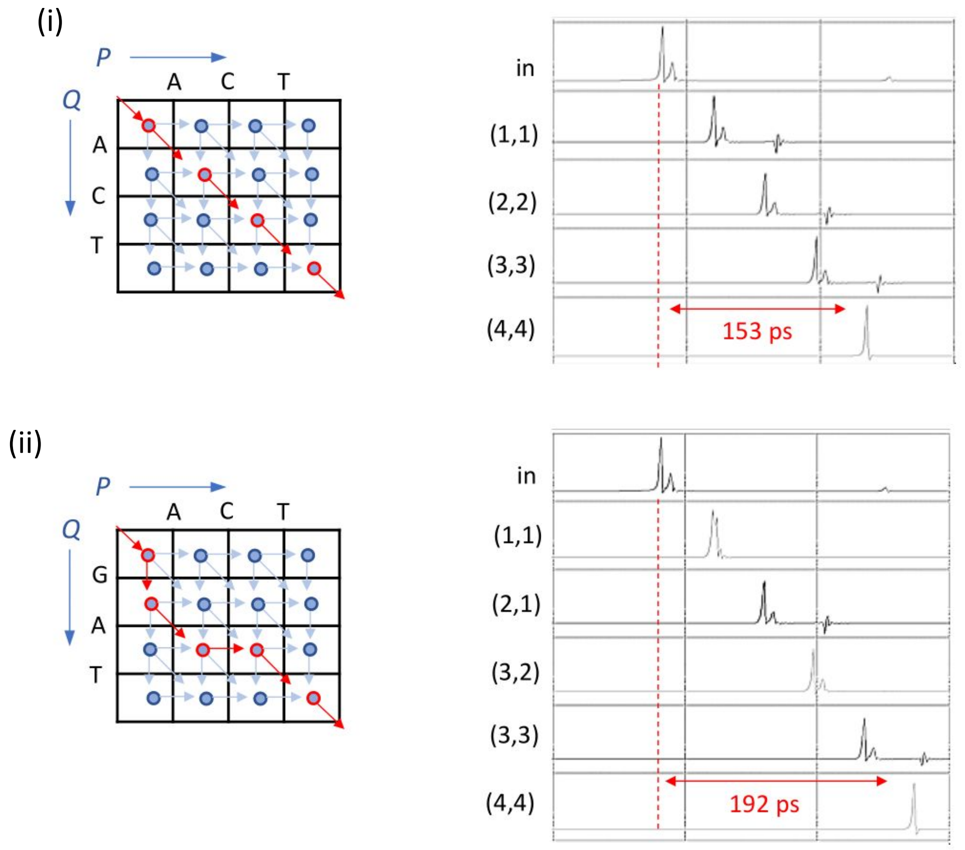


Figure 3.7: Shortest path and simulation results for a 3×3 sequence alignment problem. Panel (i): P = ACT and Q = ACT. Panel (ii): P = ACT and Q = GAT.

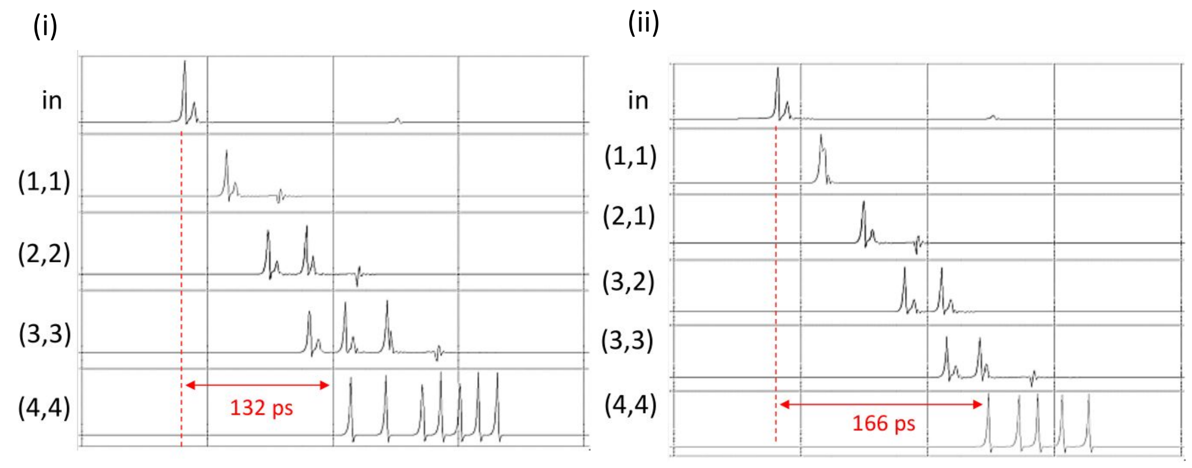


Figure 3.8: Simulation results for a stateless implementation of the sequencing accelerator depicted in Figures 3.6. Panel (i): P = ACT and Q = ACT. Panel (ii): P = ACT and Q = GAT.

unit cells. These results match our expectations, as according to our measurements, the propagation delay of each unit cell is ∼38 ps.

Note that it is possible to make these unit cells even faster and smaller by relaxing the race logic constraint for at most one event per wire per computation. This relaxation allows the implementation of a FA gate with a single merger cell. The outcome is an accelerator with fewer JJs and a ∼14% lower latency. Simulation results for the two examples discussed above are shown in Figure 3.8. The above relaxation is not always safe and should be performed with caution on a case by case basis.

Arbitrary Function Table

Our last proof-of-concept design is that of an arbitrary function table [15]. Figure 3.9 (i) provides an example case. For its implementation, synchronous Coincidence gates are used. From these Coincidence gates, we construct our temporal design's basic building block, which leverages the data-driven self-timing scheme proposed above and is shown in Figure 3.9 (ii). The resulting system is illustrated in Figure 3.9 (iii). To successfully handle time-skewed inputs, a delay $\delta = 10$ ps is introduced after each merger cell. A delay element δ' is also used to balance the delays of the two parallel paths that feed the second Coincidence gate. Both δ and δ' delays are implemented with JTLs.

Simulation results are provided in Figure 3.9 (iv). In the particular case, D1x is set to 50 ps, a is equal to 0, b is equal to 1, and c is equal to 2. As expected, a pulse appears at the output of the upper block m_0, colored in red, at $t = 209$ ps and goes through the succeeding 3-input FA gate without any interruption. Note that stateless merger cells instead of stateful FAs are used again. The total delay is equal to 219 ps. If now we subtract from this number the delay of C_3 ($\sim$70 ps), we end up with 149 ps of delay, which corresponds to the desired value of 3. No pulses come out of the two other blocks, colored in blue and green, which relate to two bottom entries of the function table. The total Josephson junction count of this design is 565 JJs.

3.6 Conclusion

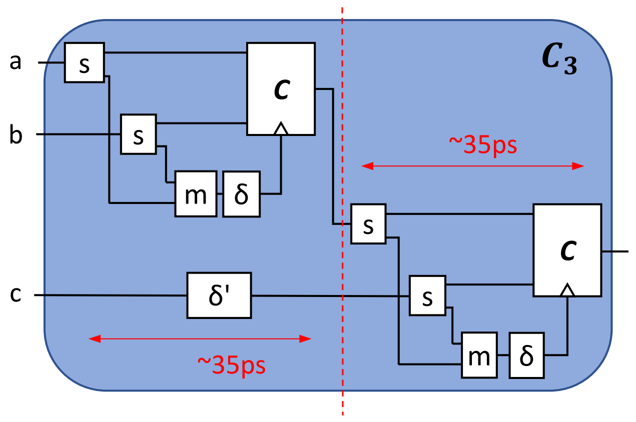
Continuous and extended effort has already carried the superconducting field from the first fabricated Josephson junction in 1970 through the development of RSFQ logic in 1985 to chips with densities on the order of several million JJs today. With the realization of self-shunted JJs in 2017 [55], chips with 10M JJs are now in sight. The excitement around quantum computing further drives the demand for improvement in superconductor circuit fabrication. However, not all superconductor limitations come from the device level. Some of the most profound result from the pulsed-based nature of computation.

In this chapter, these challenges were approached from the angle of logic design and computer architecture. By removing CMOS-oriented bias and focusing instead on the way information is encoded in superconducting, we showed the benefits of replacing strictly

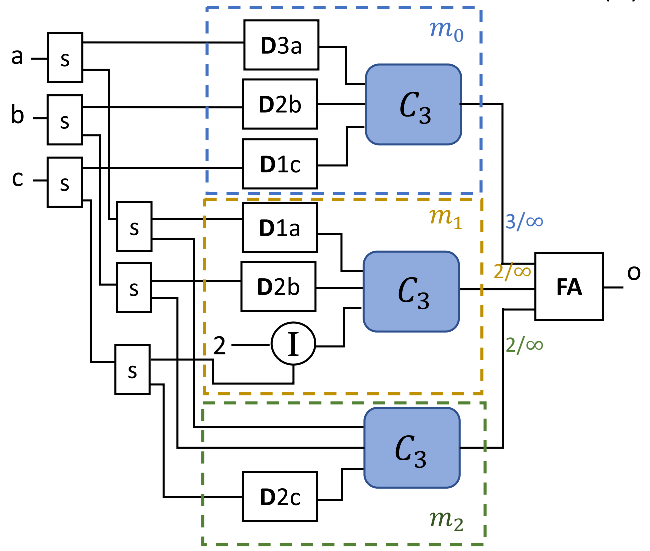
(i)

a	b	c	o
0	1	2	3
1	0	∞	2
2	2	0	2

(ii)



(iii)



(iv)

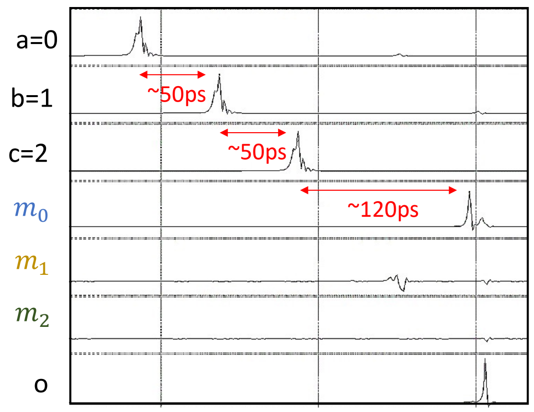


Figure 3.9: Panel (i): specification of an example function table. Panel (ii): Block diagram of a self-timed 3-input Coincidence gate. Panel (iii): Block diagram of the corresponding accelerator design. Panel (iv): WRSPICE simulation results for $a = 0$, $b = 1$, and $c = 2$.

Boolean logic with temporal abstractions. To support this approach, we presented Mealy machines that describe the functionality of temporal operators, showed how existing SFQ elementary cells can be repurposed for the clock-free implementation of these “temporal” Mealy machines, and developed superconductor accelerator designs that compute faster and more efficiently than their semiconductor counterparts, while avoiding the clocks and memories that shackle more incremental superconducting approaches.

Looking forward, we expect this approach to drive innovation and in other pulse-based technologies and serve as a blueprint for the exploration of models of computation that come closer to the innate properties of underlying novel devices. As for temporal superconducting temporal logic itself, a next step includes the systematic development of a scalable and universal resetting scheme.

Chapter 4

From Temporal to Superconductor Boolean

4.1 Introduction

In conventional superconducting approaches, the clock plays two roles. First, it provides the time interval necessary for defining logic values (see Figure 3.2). Second, it serves to reset, or "relax", a gate's state back to ground, so it is ready to receive new inputs in the next cycle. In Chapter 3, we discussed how temporal computing can provide a clock-free alternative. Here, we shift our focus back to Boolean superconductor circuits and propose a solution to the problem of constructing Boolean logic from stateful superconductor elements communicating via impulses.

Through the careful co-design of logical value encoding, SFQ circuit elements, and architecture, we are able to (a) maintain well understood and composable Boolean logic abstractions, while (b) avoiding clocks at every logic gate and the insertion of delay pads. In doing so, we (c) ensure stateful elements always return to the ground state, which (d) allows computer architecture considerations, such as pipeline depths, to drive design decisions rather than circuit-level requirements. Finally, we (e) provide physically realizable, low-energy, and low-delay implementations that are fully compatible with existing SFQ design processes. We demonstrate that these properties are achieved through four core contributions.

First, we show that AND and OR gates can be envisioned as pulse-based Last Arrival (LA) and First Arrival (FA) operations. Although LA and FA cells are asynchronous (and hence clock-free), they remain stateful. Correct operation depends on guaranteeing that the asynchronous cells are always returned to their initial state prior to the start of the next round of computation. This can be achieved by dividing a logical clock cycle into two alternating synchronous phases: excite and relax. The excite phase operates on pulse-coded logic values, while the relax phase propagates the values needed to return the asynchronous cell back to its ground state. This is essentially an implementation of alternating logic [56], in which logic state machines always return to the correct initial state at the end of each two-phase logical cycle.

Second, although LA and FA cells implement AND and OR gates, AND and OR alone are not functionally complete. To establish functional completeness, a NOT function is needed. Here, we build on the theory of unordered binary codes [57, 58]. In an unordered binary code—one in which no codeword covers another—any Boolean function can be implemented using only AND and OR gates. The classic dual-rail (DR) approach is a special case; in contrast to existing DR-based SFQ approaches [59, 60, 52, 53], which rely on synchronous logic gates and use complementary data signals to generate the required clock or control signals, in xSFQ, DR codes are used only for guaranteeing functional completeness.

Third, we revisit the superconductor implementation of LA and FA cells. The previously described SFQ implementations (Chapter 3) have imbalanced delays and hardware redundancies, which leave room for optimization. The revised cells require 30% fewer

JJs and have a propagation delay of 9 ps.

Fourth, we present alternation-aware registers along with a new pipeline balancing technique capable of hiding much of the performance overhead inherent in a two-phase approach. In order to make sequential networks amenable to alternation, each logical flip-flop in the design is implemented with a coupled pair of destructive readout (DRO) cells. The DRO cells are then distributed along a combinational logic pathway in a manner that is analogous to traditional circuit retiming [61].

To evaluate the functionality and performance of our logic system, dubbed xSFQ, we construct detailed SPICE-level models of the proposed cells, perform discrete-event simulations of more complex superconductor alternating systems, extend CMOS-oriented analytical power-performance models to fit superconducting technology, and demonstrate the benefits of xSFQ over conventional SFQ-based systems through an energy-delay product (EDP) analysis. We find that for a design resembling a RISC-V RV32I core [62] (consisting of 10,000 two-input logic gates and a critical path of 150 gates) and a 10% pipeline hazard ratio (HR), xSFQ achieves 14× EDP reduction excluding the overhead of interlock and flushing circuitry. These gains increase super linearly with the length of the critical path, the number of synchronous buffers required to equalize uneven datapaths, and the ratio of pipeline hazards. For example, for 15% and 20% HRs, 22× and 31× EDP savings are observed, respectively, excluding the overhead of interlock logic.

4.2 Current Status

Superconductor ALU designs [63, 64] and microprocessors [65, 66, 67, 68] have been presented in an effort to capitalize on the promise of superconductors. The majority of these implementations are based on simplified architectures, bit-serial processing, and shift-register-based on-chip memories. For example, the modern CORE e4 [67] is an 8 bit-serial RSFQ microprocessor that contains 4 general-purpose registers, can execute 20 different instruction types, and achieves up to 333 Million Instructions Per Second (MIPS) while dissipating an estimated 2.03 mW of power. To the best of our knowledge, this estimate does not include the cost of cooling, which increases power requirements by approximately two orders of magnitude according to Carnot's thermodynamic efficiency theorem [69].

More recently, there has been increasing interest in the exploration of superconductor accelerators for emerging applications. For example, Tannu et al. [70] developed a reciprocal quantum logic (RQL)-based design for SHA-256 (cryptographic hashing) engines [71]. To maximize their gains, the authors focused on the optimization of adder circuits, which are the most critical components of the SHA engine, and proposed a fault-tolerant architecture that allows JJ critical currents to be lowered from 38 μA to 10 μA. The reported results indicate 46× energy efficiency gains and 20% performance gains compared to CMOS. Ishida et al. [72] presented an ERSFQ-based neural processing unit. Based on the reported results, the performance per Watt of the proposed design is 490× and 1.23× higher than a tensor processing unit (TPU)-like [73] CMOS implementation, without and with the cost of cooling accounted for, respectively.

Other interesting approaches that target accelerators rely on the exploitation of less traditional computing paradigms that match well with the characteristics of superconductor electronics. For example, Cai et al. [74] presented a deep learning framework based on stochastic computing that uses adiabatic quantum-flux-parametron (AQFP) technology. According to their simulation results, the proposed deep neural network (DNN) design can achieve up to 6.9×10^4 times higher energy efficiency than CMOS—-to the best of our knowledge, this again excludes the cost of cooling. Another approach that falls under this category is the one presented in Chapter 3. We claimed that the natural language for expressing computations in a pulse-based system is one that precisely describes the temporal relationships between these pulses. To support this argument, we first implemented the four basic temporal operators in SFQ, then we set the foundation for the design of more complex temporal superconductor circuits, and finally provided a proof-of-concept decision tree accelerator design, achieving $\sim 40\times$ lower latency compared to CMOS (with more examples shown in the original papers [8, 9]).

These ideas pave a promising path forward. Nevertheless, their shortcomings and underlying assumptions cannot be ignored. The computational limits and efficiency of stochastic computing and temporal logic for general-purpose tasks have yet to be explored, especially for superconductor systems. The remaining accelerator architectures support only dataflow processing without complex control flows, while the demonstrated microprocessors (and other designs) are not scalable. For example, several existing benchmark SFQ circuits require a significant number of DRO cells for padding uneven datapaths. According to published results for the ISCAS85 benchmark circuits [75], the number of required DRO cells for padding exceeds the number of logic gates by more than $2.5\times$ on average (ranging from $1.5\times$ to $5\times$) [76]. Furthermore, the only realistic way to avoid extremely high logical cycles per instruction (LCPIs), at least in the case of microprocessors, is to apply fine grained temporal multithreading [77] where, ideally, the number of threads is as large as the number of gates on the critical path. Additionally, the throughput commonly reported in superconducting papers refers to a theoretical peak rate based solely on the frequency of individually clocked gates (e.g., delays due to pipeline stalls are excluded). Every gate is essentially a pipeline stage, and extremely deep pipelines put more pressure on the already challenging superconductor memory system.

4.3 xSFQ Logic Design

At the physical level, conventional CMOS and superconductor technologies are radically different. Yet, it is obviously desirable if the tools, techniques, and computer architecture concepts developed for conventional computer systems can be applied to superconductor designs. We strive to achieve this at the logic level of abstraction. In other words, we aim to first perform digital design using what appear to be ordinary logic gates and then map them in a straightforward way to superconductor cells.

The differences between conventional and superconductor technologies, however, introduce an entirely different design space, with a distinct set of trade-offs and constraints. As discussed above, one of the most important of these involves cell fan-outs. Superconductor cells by themselves are limited to driving one cell. Fan-outs of two or more require splitters, which are limited to a fan-out of two. Hence, large fan-outs must be constructed as binary trees of splitters. This means that clock, global reset, and other high fan-out signals are relatively expensive compared to CMOS.

Regarding fan-in, although it is possible to design superconductor cells with more than two data inputs, they are not commonly used because they are significantly more complex and mostly developed in an ad hoc way. Hence, for our purposes, it is assumed that superconductor cells are limited to two inputs. Of course, in the logic domain, the designer may use multi-fan-in gates, with the awareness that these will be expanded into expensive multi-cell superconductor circuits.

4.3.1 Clocking Discipline

In keeping with the above design philosophy, we propose and develop a clocking discipline that is aligned with conventional synchronous CMOS; e.g., ranks of clocked storage elements separated by unclocked combinational networks. This is in sharp contrast to existing superconducting methods where, typically, every gate is clocked.

In the logic domain, these basic clocked storage elements operate as (non-latching) D flip-flops (DFFs) that translate to destructive readout (DRO) cells in the xSFQ domain. Figure 4.1 provides a Mealy machine-based representation of a DRO cell. Recalling that a logical clock cycle consists of two synchronous phases, at the logic network level, a synchronizing phase pulse causes the stored signal pulses to be released from the DRO cells. Then, signal pulses propagate through a forward flow network of asynchronous xSFQ cells with pulses eventually arriving at downstream DRO cell inputs. The subsequent phase releases them from that stage as the process continues. As with conventional synchronous logic design, the phase period must be long enough to allow for propagation along the longest signal path through the xSFQ combinational network (and respect setup time requirements).

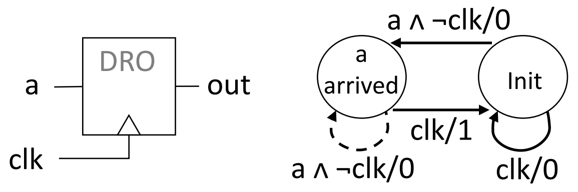


Figure 4.1: Symbol and Mealy machine representation of a Destructive Read Out (DRO) cell.

As part of the logical signaling discipline, a given data wire transmits at most one pulse during a given (synchronous) phase. The timing of the pulse within the phase period does not affect its logical interpretation—only its presence or absence matters. Hence, the designer should worry only about the timing constraints set by the lengths of signal paths, which must fit within a clock phase, and not timing relationships across paths.

4.3.2 First Arrival and Last Arrival Cells Specification

According to their semantics [8, 9], FA and LA cells implement asynchronous (unclocked) state machines that operate on pulsed inputs and produce pulsed outputs. As its name indicates, a FA cell emits an output pulse in response to the first input pulse that arrives at either of its inputs. Any later input pulse does not affect the FA cell output (see Figure 4.3 (i)). A LA cell emits an output pulse only if both inputs receive pulses. The output pulse occurs in response to the second input pulse ((see Figure 4.3 (ii)).

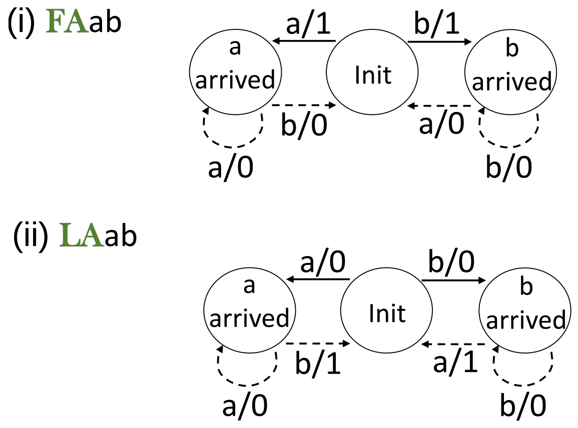


Figure 4.2: Description of (i) FA and (ii) LA cells' functionality with Mealy machines.

Figure 4.3 illustrates the state transitions of these cells as they occur during the time frame of a (clock) phase (although the cells are not clocked themselves).

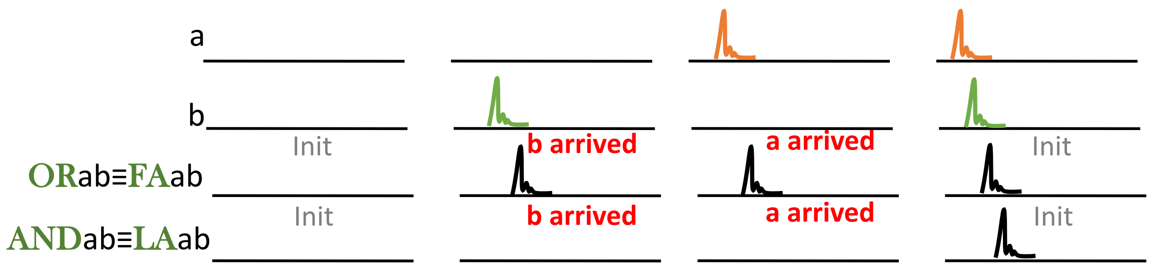


Figure 4.3: State transition tables and output responses for FA and LA cells.

When used in this way, we observe that the FA cell implements an OR function with respect to pulse signals and the LA cell implements an AND function. For proper operation of both cell types, the initial state must be *Init* and each input line must be restricted to one pulse per phase. The arrival of more than one pulse indicates a violation of the signaling convention.

4.3.3 Two-phase Alternating Encoding

Based on the shown Mealy machines alone, there is no guarantee that the initial state will be restored at the end of a phase. For example, if the FA cell observes only a single input pulse during a given phase, it will be in either the *a arrived* or *b arrived* state at the end of the phase. To force FA cells to transition back to the *Init* state, the obvious solution is to fan out an explicit global reset signal to all FA cells. The same holds true for LA cells. But, as in the case of fine-grained clocking, this requires a binary tree of active splitters and will be very expensive.

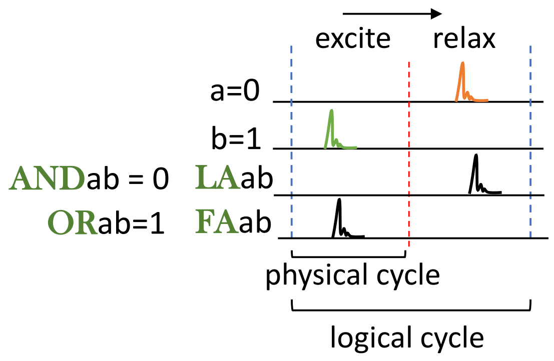


Figure 4.4: The value of an alternating binary variable appears during the excite phase and is followed by its complemented value during the relax phase. Each phase corresponds to a physical cycle (no two-phase clocking is happening). An excite-relax pair forms an xSFQ logical cycle.

We propose a novel approach that accomplishes a state reset by using only the functional input wires. There are no explicit reset wires, and thus no need for a reset distribution network. This approach does so by providing all mechanisms to reset the gates within the logic encoding itself. This is the motivation for dividing a logical cycle into a pair of physical cycles, or synchronous phases, where an excite phase is followed by a relax phase. In the excite phase, the pulsed inputs are given their logically equivalent values. During the relax phase, the pulsed inputs are given their complement values (see Figure 4.4). This guarantees that each FA and LA cell always receives exactly one pulse at each of their input ports throughout the two-phase logical cycle and returns its initial state. We define this as an alternating encoding.

Figure 4.5 presents all possible alternating input pulse sequences for FA and LA cells. Both the excite and relax phases are shown for all legal input combinations. In every case, if the initial state is *Init* and the input pulses alternate between the excite and relax phases, then (a) the output during the relax phase is opposite the output for the excite phase, and (b) the final state is always *Init*. We define this as the alternating signal property.

	excite					relax				
state	**inputs**		**FA**ab	**LA**ab	**state**	**inputs**		**FA**ab	**LA**ab	**state**
	a	b				a	b			
Init	0	0	0	0	Init	1	1	1	1	Init
Init	0	1	1	0	b arrived	1	0	0	1	Init
Init	1	0	1	0	a arrived	0	1	0	1	Init
Init	1	1	1	1	Init	0	0	0	0	Init

Figure 4.5: Alternating input pulse sequences for FA and LA cells.

Importantly, the alternating signal property holds not only for individual cells, as just shown exhaustively, but also for any network composed of these cells.

Theorem 1 *Any feed-forward network composed of FA and LA cells will have the alternating signal property.*

Proof sketch. An individual FA or LA cell is alternating, and is a network of depth 1.

If two networks having the alternating signal property connect to the inputs of an FA or LA cell, then that FA or LA cell must observe alternating inputs, so its output satisfies the alternating signal property with respect to the network inputs. By induction, the overall network is therefore alternating [1].

Hence, by using alternating signal inputs (which consume two synchronous phases), a state reset is achieved without an explicit reset signal. Clearly the two phase clocking system requires additional time compared to a single phase system, but an explicit reset signal would also consume additional time, perhaps as much as a full clock phase. Additionally, the total number of pulses over an excite-relax pair is always constant, which may be useful for the detection of erroneous operation [56].

4.3.4 Unordered Codes and Functional Completeness

As just described, an FA cell implements a logical OR gate that operates on pulses, and an LA cell implements a logical AND gate. Nevertheless, a NOT gate is missing. Implementing a NOT gate is typically problematic with pulse signaling because it implies knowledge that a pulse will not occur during the leading excite phase, but a circuit cannot wait until this phase is over to make this determination, and it cannot look into the future. However, if input and output signals are constrained to be members of an unordered code, then functional completeness is achievable with AND and OR gates alone.

Definition 4.3.1 *A vector $X = x_1x_2...x_n$ is said to* cover *another vector $Y = y_1y_2, ...y_n$ (denoted $X \geq Y$) if for all $i \in n$, $y_i = 1$ implies $x_i = 1$. Vectors X and Y are unordered if $X \ngeq Y$ and $Y \ngeq X$.*

For example, if $X_1 = [0011]$ and $Y_1 = [0111]$, Y_1 covers X_1. On the other hand, the vectors $X_2 = [1001]$ and $Y_2 = [1010]$ are unordered.

[1] Splitters are used for fan-out, and thus are not considered logic elements.

Definition 4.3.2 *A binary code is unordered if no two members of the code are ordered.*

Well-known examples of unordered codes include one-hot codes, dual-rail (DR) codes, Berger codes [78], bi-quinary codes—for example, the IBM 650 was a bi-quinary coded decimal computer [79])—and various ad hoc codes [57, 58].

Theorem 2 *Any Boolean function whose domain consists of an unordered code can be implemented using only AND and OR gates.*

Proof sketch. By construction—the complement of any bit can be formed as an AND/OR function of the other bits. This is done by first selecting all codewords for which the subject bit is a 0, and then forming a set of minterms corresponding to the 1 bits in the selected codewords. Summing the minterms yields the complement of the subject bit.

Example: Consider the 2-out-of-4 code (each tuple consists of 4 bits and has exactly 2 logical 1s), $[x_1x_2x_3x_4] = \{[1100], [1010], [1001], [0110], [0101], [0011]\}$. The value of $\overline{x_1} = x_2x_3 + x_2x_4 + x_3x_4$.

In practice, arbitrary unordered codes are difficult to work with. Although the complement of any bit can be formed via an AND/OR function of the other bits, the AND/OR network can become quite large. Even for the relatively small 2-out-of-4 code presented above, the complement of any bit requires 3 AND gates and 2 OR gates, all of which have a fan-in of 2.

Generally speaking, the more efficient the code, the more difficult it is to form complements. If efficiency is defined as the total number of information bits per number of code bits, the most efficient unordered code is k-out-of-$2k$. For larger values of k, k-out-of-$2k$ codes are cumbersome for implementing arbitrary functions. However, the special case k=1 yields the DR code. Although the DR code is relatively inefficient, it is particularly

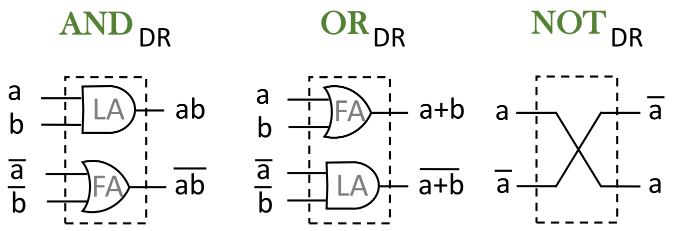


Figure 4.6: Dual-rail (DR) implementation of AND, OR, NOT functions. Each AND and OR gate consists of one FA-LA pair, just with different wiring. The NOT gate has no overhead.

simple and easy to work with. Figure 4.6 illustrates a straightforward mapping from an arbitrary logic network consisting of AND, OR, and NOT gates to a DR implementation using only ANDs and ORs. The complement of any bit is immediately available at zero circuit cost.

Corollary 2.1 *Any Boolean function having DR inputs can be implemented using only AND and OR gates.*

DR logic forms the backbone of the presented design methodology, with other unordered codes being used in cases where they lead to fewer gates or are otherwise advantageous. For example, 1-out-of-n codes are natural for holding decoded values. So, one might simply maintain some values in decoded form. Translating between a DR code and a 1-out-of-n code can be done using n gates (AND gates for DR to 1-out-of-n; OR gates for 1-out-of-n to DR).

Note that the rote mapping between AND/OR/NOT Boolean functions and DR equivalents shown in Figure 4.6 is not required. In some cases, a cheaper DR design may be achieved by other means.

Example: If one begins with the "standard" full adder composed of 9 NAND gates, the straightforward DR translation consumes 18 FA/LA cells. However, a 14 cell implementation is possible [80] if freed from using only AND and OR DR pairs (see Figure 4.7).

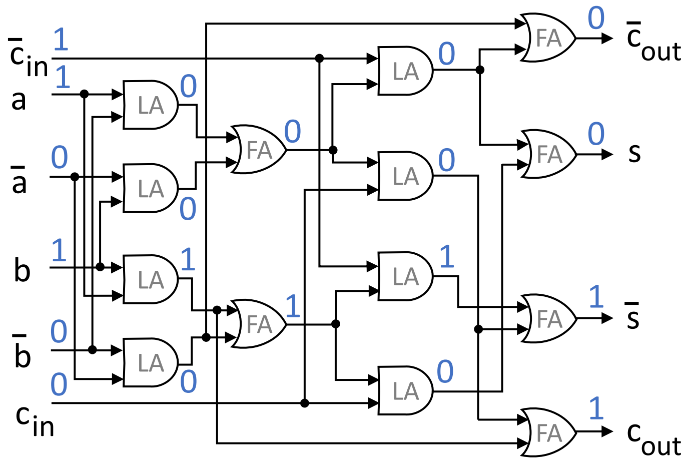


Figure 4.7: Dual-rail full adder composed of 14 FA/LA cells [80]. Example where $a = 1$, $b = 1$, and $c_{in} = 0$ is shown.

We note that inputs and outputs are still in DR format and that each logical AND and OR gate shown can be implemented with LA and FA cells, as already discussed.

4.4 xSFQ Implementation

Experimental setup. For the development and evaluation of our analog models, we use Cadence's Spectre simulator and superconductor device model files for the SFQ5ee process [47].

4.4.1 First Arrival and Last Arrival Cell Implementations

Our goal is to design circuits that satisfy functional correctness and: (a) evaluate without the need for a clock signal, (b) return to the ground state at the end of a logical cycle without the need for explicit wired reset signals, (c) provide a practical way to reinitialize in case an error occurs due to faulty hardware (or for any other reason), (d) achieve relatively similar propagation delays on datapaths that are at least as short as their SFQ AND/OR counterparts, and (e) minimize the JJ count, which affects area and defines the power dissipation of the circuit.

For the design of the LA circuit, shown in Figure 4.8 (i), we use as a reference the clockless dynamic SFQ AND gate originally developed by Rylov [51]. As already discussed, superconducting cell states arise from the storage of flux within SQUIDs (loops formed by two JJs, one on each side of a quantizing inductor). Here, such loops are formed around the input inductors L0 and L1, the output JJs from the prior stage, and J4. The output JJs from the prior stage are not shown, but are built into every gate with an output wire. In the top loop, L0 holds input a until input b arrives; likewise, in the bottom loop, L1 holds b until a arrives. Both inputs must arrive for the circuit to emit an output pulse and properly reset. In xSFQ, one pulse will arrive at each input port during a two-phase logical cycle, and thus the LA cell will transition from the *Init* state to either the *a arrived* or *b arrived* state and back to *Init* in one full cycle. If, for some reason, the second input does not arrive before the end of the logical cycle, the stored flux is removed by the J0-J2-R0 or J1-J3-R1 loop using a technique referred to here as bleed-out because it drains the flux quantum over time. The bleed-out rate of the cell depends to some extent on the amount of serial resistance. In the provided example, the bleed-out window is set to 29 ps, for which we used an R0 value of 0.67 Ω. However, when R0 is set to 0.70 Ω, the bleed-out window increases to 42 ps. Note that in all shown cases but the last, a and b input pulses arrive within a 29 ps offset (which is the cell's bleed-out window); thus, there is an output pulse. In the last case, the offset is 58 ps and no output pulse appears.

Figure 4.8 (ii) shows the schematic of the FA cell. It is implemented with a modification of an inverted C-element, originally presented by SUNY researchers [43]. Each input has a SQUID that stores the opposite input's flux until both signals have arrived—that is, when input b (a) arrives, it is both propagated to J4, generating an output pulse, and to SQUID J0-L0-L1-J1 (J2-L2-L3-J1), which stores the flux until a (b) arrives and

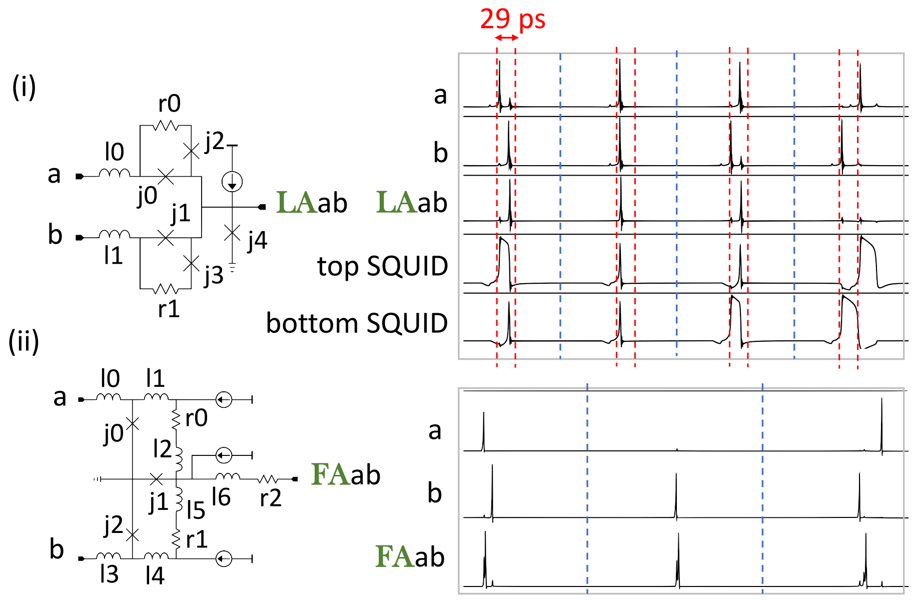


Figure 4.8: Panel (i): LA cell schematic and waveform. The simulations cover both cases where input pulses arrive within and outside the bleed-out window boundaries. The results shown provide evidence that the cell satisfies the alternating logic requirements (used for normal operation) and supports "bleed-out" (used to recover from a faulty operation). For this example, the bleed-out rate, which is how quickly the SQUIDs can drain their respective fluxes, is 29 ps. Panel (ii): FA cell schematic and waveform. In the first and third cases, both input pulses arrive. The cell fires upon the arrival of the first input pulse, transitions to the *a arrived* or *b arrived* state, and waits for the second input to return to the *Init* state. In the second case, only one of the input pulses arrives (faulty operation). The bleed-out feature allows the gate to return back to the *Init* state.

cancels it out. Similar to the LA cell that bleeds out the flux, this cell also bleeds out through small serial resistors R0 and R1. We add a current source to the central node to mitigate bias current redistribution, and remove several redundant JJs on the input and output wires to reduce propagation delay.

Note that even in the rare case where two input pulses arrive simultaneously, both LA and FA cells behave as expected (there are no race conditions). Also, because clocking is not part of their semantics, the cells are delay-insensitive. Additionally, their bleed-out feature is used for resetting only in the case of faulty operation. Thus, when designing an xSFQ system, the bleed-out window of LA/FA cells should be configured to the length of a logical cycle (two synchronous phases).

Implementation results are summarized in Table 4.1. To estimate the energy consumption of the designs, we assume that all JJs switch over a logical cycle. The switching energy of a single JJ is 2×10^{-19}J. Compared to prior LA and FA implementations (Chapter 3), the presented FA and LA cells support bleed-out, have more balanced propagation times, and require fewer JJs. The bias margins for both elements, according to our experiments, are -30% to +30%.

Table 4.1: Area, latency, and energy estimates of LA and FA cells implemented in the SFQ5ee process.

Element	Area (#JJs)	Latency (ps)	Energy (aJ)
LA	5	8	1.0
FA	3	9	0.6

In comparison to more traditional SFQ implementations of Boolean AND and OR gates (Table 4.2), the xSFQ-based implementations require at least 30% fewer JJs and 55% less energy than their counterparts, even when LA and FA cells are used in pairs. Regarding latency, although 50% longer delays may be observed in single 2-input gates due to the two-phase nature of the proposed DR alternating logic scheme, for composite functions, xSFQ designs still deliver performance gains. These gains are expected to grow with the size of the design, as the timing overhead incurred by fine-grained clocking is no longer a consideration.

Table 4.2: Area, latency, and energy estimates of synchronous SFQ AND and OR circuits implemented in the SFQ5ee process.

Element	Area (#JJs)	Latency (ps)	Energy (aJ)
sync. AND	11	9	2.2
sync. OR	12	8	2.4

To provide further evidence of xSFQ circuit efficiency, we use an 8-bit ALU as a reference example. An SFQ implementation [81] consists of 9 pipeline stages (fixed and equal to the number of gates on the critical path), requires 4,908 JJs, and achieves 120 TOPS/W in low-voltage RSFQ (1.4 POPS/W if only the switching power is considered, $\sim$ 22 μW). In xSFQ, pipeline depth is configurable. For a purely combinational design, our estimates are: 2,800 JJs, 1.7 μW switching power, and 1.8 POPS/W. Note that in the above analysis, we assume that all pipeline stages in SFQ will always be busy.

4.4.2 Storage Elements

The above-described FA and LA cells, along with the alternating DR encoding introduced by xSFQ, are sufficient for the implementation of any combinational logic block. However, for the realization of real-world computing systems, xSFQ-amenable storage elements are still needed. As already discussed, in xSFQ, each logical cycle consists of a pair of physical cycles (excite-relax phases). Thus, an xSFQ storage element must be able to generate the time-offset complement with respect to the primary excite phase.

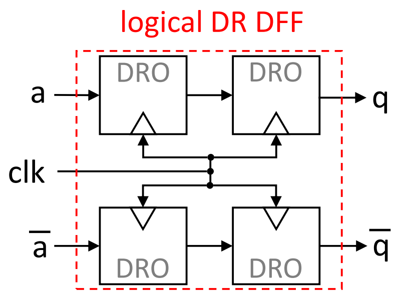


Figure 4.9: xSFQ DR latch built from DROs. To fan-out the clock signal, we use three splitters, which are signified by black dots.

A simple method of implementing an xSFQ latch is to use a pair of synchronous DRO cells. Figure 4.9 illustrates such a component for the case of DR codes. In particular, the shown circuit latches the a or $\overline{a}$ when it arrives on the excite phase and releases it at the start of the next excite phase. Likewise, the data latched in the relax phase are written out on the next relax phase. Implementation results for DRO and splitter cells are in Table 4.3.

Table 4.3: Area, latency, and energy estimates of DRO and splitter circuits implemented in the SFQ5ee process [47]

Element	Area (#JJs)	Latency (ps)	Energy (aJ)
DRO	6	5.1	1.2
Splitter	3	4.3	0.6

Note that although xSFQ requires data in a DR alternating format, traditional binary storage is still possible with the use of binary-to-alternating-DR (BAC) and alternating-DR-to-binary (ABC) converters. Example BAC and ABC designs are shown in Figure 4.10.

4.5 Phase Rebalancing

With asynchronous combinational logic elements and synchronous storage elements available, one can perform digital design in xSFQ just as in CMOS without being constrained by pipelining with gate-level granularity. Figure 4.11 (i) illustrates a pipelined xSFQ circuit, where a block of combinational logic is surrounded by synchronous register blocks.

Each logical DFF is implemented in xSFQ as a “double-pumped” latch (Figure 4.9) and the combinational logic consists of interconnected LA and FA cells. Although this structure is fully functional, if we strip each logical DFF down to the very basic digital design equivalents, the resulting system is unsatisfying from an architectural standpoint: it is a completely unbalanced pipeline, because no computation is done between two successive clock-synchronous DRO cells.

Observe that even though each pair of DRO cells is part of the same logical DFF, the DRO cells can be split and redistributed in a balanced way through retiming [61]. Figure 4.11 (ii) depicts a rebalanced version of the circuit shown in Figure 4.11 (i). As

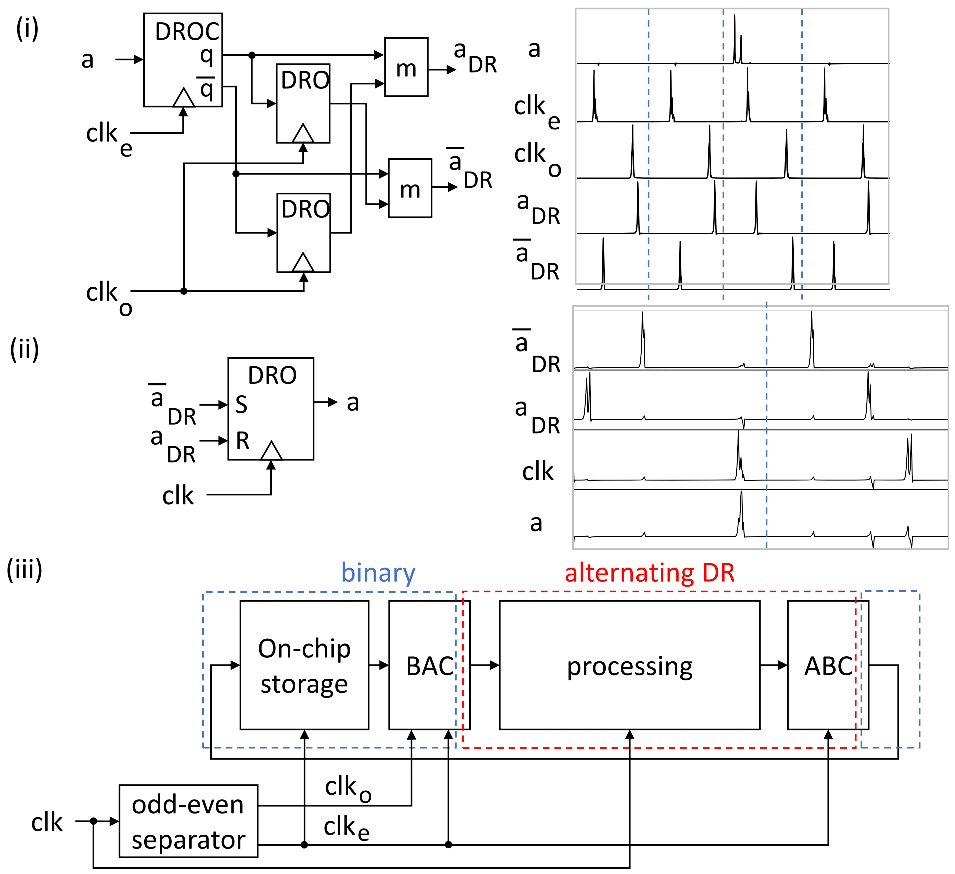


Figure 4.10: Panel (i): Binary-to-alternating-DR converter (BAC) block diagram and waveform. Panel (ii): Alternating-DR-to-binary converter (ABC) block diagram and waveform. Panel (iii): System-level view.

the DRO cells are pushed through the fabric of combinational logic, the excite and relax phases become balanced. In the ideal case, where the DRO cell propagation delay is zero and the combinational logic is perfectly balanced, retiming can completely hide the overhead associated with the relax phase (see Figure 4.11 (iii)).

Functional evaluation. In the above sections, we presented SPICE-level models and simulation results for all xSFQ logic elements. For the evaluation of more complex systems, we develop and open source[2] PyLSE [11], a discrete-event simulation framework for pulse-based systems. To effectively express the behavior of the primary elements, we

[2]https://github.com/UCSBarchlab/PyLSE

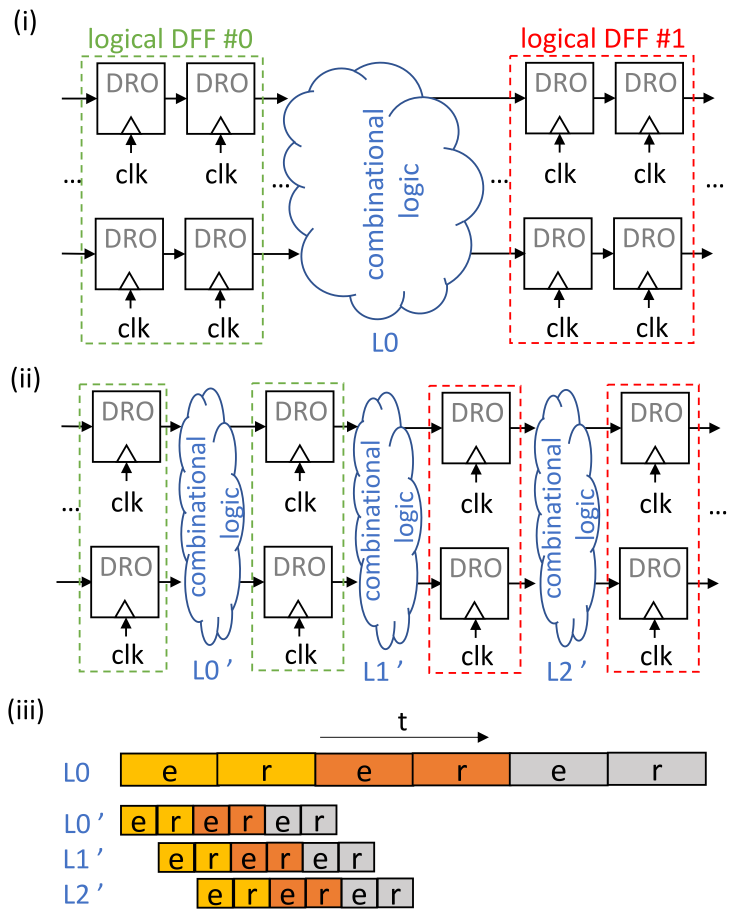


Figure 4.11: Panel (i): Unbalanced xSFQ pipeline. Panel (ii): Distributing DRO cells in a balanced way through combinational logic blocks via retiming; L0 ≡ L0'+L1'+L2'. Panel (iii): Under the assumptions of zero DRO propagation delay and perfectly balanced retiming, 100% of the delay overhead introduced by alternating encoding can be hidden. We use letter *e* to represent excite phases and letter *r* for the relax phases. An excite-relax pair forms one logical cycle.

opt for a lightweight, object-oriented state machine-based implementation. The state machine corresponding to each cell contains all legal state transitions and allows for the easy diagnosis of fan-out violations or logical faults (e.g., fewer or more than the number of expected pulses appearing on a line). Moreover, each element object can store the number of JJs required for its physical implementation and its propagation delay, which facilitates area estimation, timing analysis, and a more realistic simulation. Events are discrete variables, not continuous ones, and simulation is based on the event-oriented paradigm, in which all pending events are first stored as a set and then inspected based on their scheduled event times.

To provide evidence for the functional correctness of the proposed retiming methodology, we apply it to the full adder design shown in Figure 4.7. Figure 4.12 provides simulation results for an unbalanced circuit—a DR full adder implemented as a combinational circuit and followed by a rank of logical DFFs. To make the design even more realistic, we assume that the inputs and outputs of each cell are buffered by Josephson Transmission Lines (JTLs), which are not computationally necessary but commonly used to improve flux transmission between SFQ logic cells. According to our SPICE simulations, the propagation delay of a JTL is 5.7 ps. The DRO cell setup time requirement, 2.3 ps, is also taken into account. Under these assumptions, the longest estimated propagation delay, and thus the shortest duration of a physical cycle (phase), is 72 ps.

The rebalanced full adder design is in Figure 4.13. In this case, the critical path consists of 2 xSFQ cells instead of 4. The longest propagation delay is 44.4 ps (the duration T of a physical cycle/phase is set to 45 ps in our simulation) and the number of logical cycles required to complete computation remains the same.

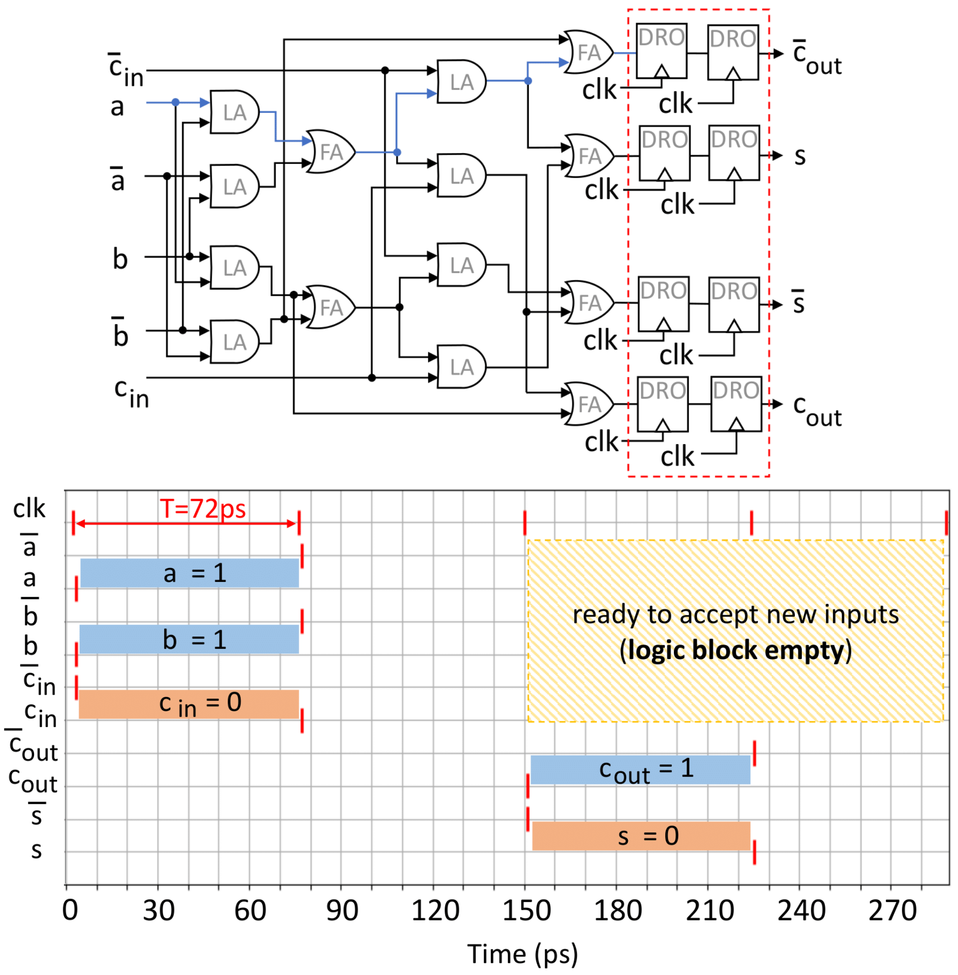


Figure 4.12: Simulation of the DR full adder design shown in Figure 4.7. Vertical red lines (in the simulation graph) represent pulses. The minimum duration (T) of a physical cycle (phase) is defined by the delay of the longest signal path through the xSFQ combinational network (colored blue). For easier reading, we also illustrate the logical values of the circuit's input and output variables.

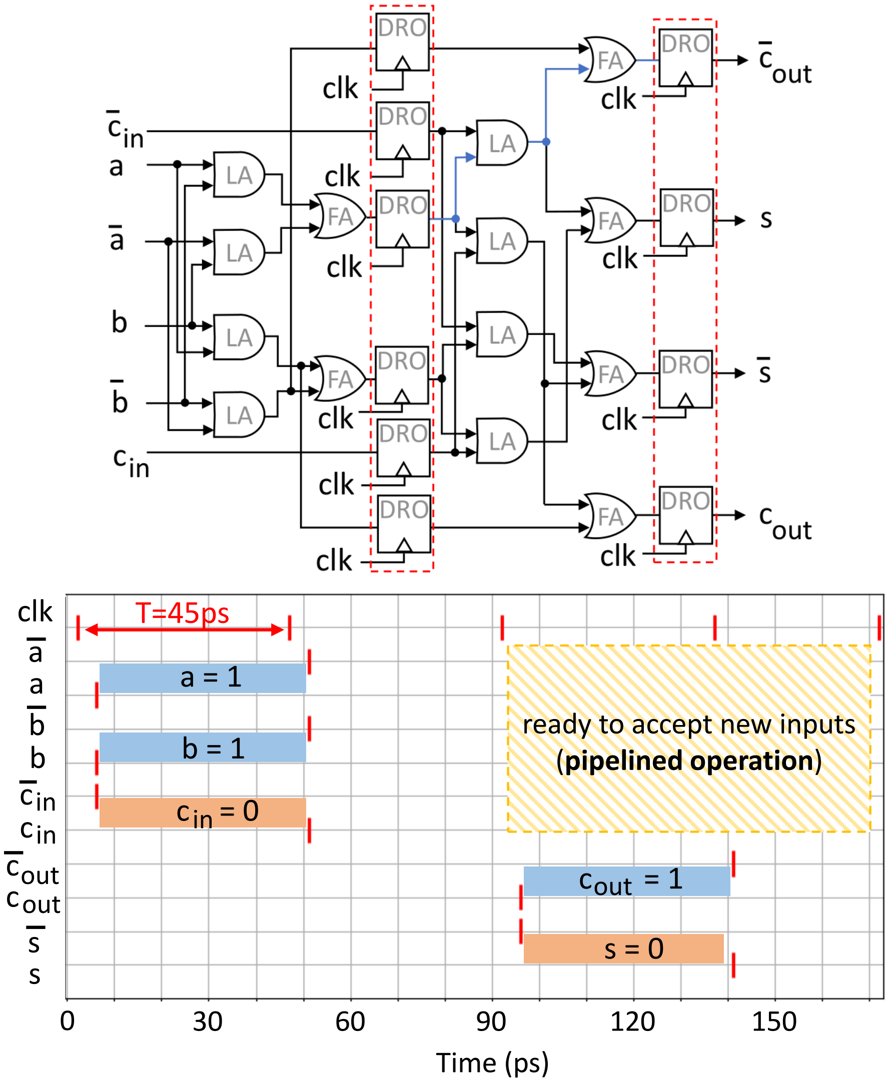


Figure 4.13: Rebalanced DR full adder design and simulation. Vertical red lines in the simulation graph represent pulses. The minimum duration (T) of a physical cycle (phase) is defined by the delay of the longest signal path through the xSFQ combinational network—colored blue in this simulation. For easier reading, we also illustrate the logical values of the circuit's input and output variables.

4.6 Optimum Pipeline Depth

Finding the optimal pipeline depth for a microprocessor is probably one of the most well-studied problems in computer microarchitecture [82, 83, 84, 85, 86]. To the first order, pipelining can offer a speed up of N when N pipeline stages are used. However, this improvement comes at the expense of dynamic power. Thus, performance and power act in opposition.

Unlike in CMOS design, architects have minimal control of the pipeline structure in prior conventional SFQ-based superconductor microprocessors [67, 77]. As already discussed, conventional superconducting Boolean gates are synchronous and act as independent pipeline stages—with the attendant benefits and problems. Considering individually clocked gates in conventional SFQ and the complexity of modern microarchitectures, we expect the number of pipeline stages in a prior conventional superconductor microprocessor to be in the order of hundreds. For example, our synthesis results of a single-cycle RV32I [62] indicate that the number of two-input gates on its critical path is approximately 150. Another noteworthy problem with the conventional approach is that the number of stages is not known until the synthesis process completes.

In contrast, the proposed xSFQ does not impose such constraints. In this section, we first revisit the CMOS performance models presented by Hartstein and Puzak [86], modify them where needed, describe the area and energy consumption of xSFQ and conventional SFQ designs as functions of the pipeline depth, and, finally conduct an energy-delay product comparison. To the best of our knowledge, this is the first study on optimal pipeline depth for a superconductor microprocessor.

4.6.1 Performance Model

Following Hartstein and Puzak [86], the basic performance metric is time per instruction (TPI), which is the inverse of the (million) instructions per second metric. TPI can be thought of as the sum of the time that a microprocessor is busy doing useful work (T_{BZ}) and the time that it is stalled because of pipeline hazards (T_{NBZ}) divided by the total number of program instructions (N_I). To calculate the number of logical cycles per instruction (LCPI) [3], which is another useful performance metric, we divide the sum of T_{BZ} and T_{NBZ} by T_{BZ}. Expressions for T_{BZ}, T_{NBZ}, TPI, and LCPI (for a scalar machine) are in Table 4.4.

We follow the original Hartstein and Puzak notation where possible. In particular, variable t_p represents the total logic delay of the microprocessor, t_s the physical cycle time, p the number of architectural pipeline stages, t_o the latch delay overhead, γ the weighted average of the fraction of the pipeline stalled by hazards ($\gamma \in [0, 1]$), and N_H the number of pipeline hazards.

In the case of xSFQ, p can take values ranging from 1 to $N_{lg\_cp}/2$, where N_{lg_cp} is the number of gates on the critical path. Exceeding this upper bound is not useful, as rebalancing no longer applies. In the t_s, T_{BZ}, and T_{NBZ} expressions, we divide or multiply by a factor of 2 to account for the effects of rebalancing. In conventional SFQ, t_s is equal to the longest propagation delay among the available synchronous SFQ gates, given that p is not a free variable. Note that the given equations do not capture timing skews or the propagation delay of interconnect lines.

[3]We use LCPI instead of cycles per instruction (CPI) because each xSFQ logical cycle consists of two physical cycles.

4.6.2 Area Model

Regarding area, we assume that the majority of hardware resources is associated with logic gates (LA, FA, and conventional SFQ Boolean cells), latches (DRO cells), and splitters. We categorize these cells as follows: LA and FA cells are considered asynchronous, and conventional SFQ Boolean cells and DRO cells are considered synchronous. As for splitters, we count only those that are part of the clock distribution network, which dominates the total splitter count. Table 4.4 provides a description of our analytical area models (in terms of gate count).

As previously discussed, in conventional SFQ, DRO cells are needed for padding unequal datapaths. We assume that their number N_{dro} increases linearly with the number of logic gates (N_{lg}); o_{dp} is the growth rate associated with the padding overhead. The number of logic cells (N_{lc}) is equal to the number of logic gates (in SFQ), and the number of splitters on the clock line (N_{splt}) matches the total number of synchronous cells; $N_{dro} + N_{lg}$ in this case.

To estimate N_{dro} in an xSFQ design, the methodology introduced by Srinivasan et al. [85] is followed. N_L is the number of logical latches for a single-stage pipeline, η is the latch growth exponent, and a factor of 2 is compensation for the additional cost of the xSFQ double-pumped architectural latches. Variable o_{dr} denotes the overhead introduced by DR codes. More specifically, if LA and FA cells are used in pairs, $o_{dr} = 2$; otherwise, $o_{dr} = 1$. This overhead is accounted for in the logic cells estimation, as well ($N_{lc} = N_{lg} o_{dr}$). Moreover, in xSFQ, the only synchronous components are DRO cells, and thus $N_{splt} = N_{dro}$.

Table 4.4: Performance, area, and energy models for xSFQ and conventional SFQ.

	xSFQ	**conv. SFQ**
$p \in$	$[1, N_{lg\_cp}/2]$	$[N_{lg\_cp}]$
$\delta \in$	$[1]$	$[0, 1]$

Performance

$t_s =$	$t_o + t_p/(2p)$	t_o
$T_{BZ} =$	$2t_s N_I$	$t_s N_I$
$T_{NBZ} =$	$\gamma N_H(t_o(2p) + t_p)$	$\gamma N_H t_o p$
$TPI =$	$(T_{BZ} + T_{NBZ})/N_I$	$(T_{BZ} + T_{NBZ})/N_I$
$LCPI =$	$1 + T_{NBZ}/T_{BZ}$	$1 + T_{NBZ}/T_{BZ}$

Area

$N_{dro} =$	$2N_L p^\eta o_{dr}$	$N_{lg} o_{dp}$
$N_{lc} =$	$N_{lg} o_{dr}$	N_{lg}
$N_{splt} =$	N_{dro}	$N_{dro} + N_{lg}$

Energy

$E_{ac} =$	$(N_{lc} e_{lc} \delta)LCPI$	0
$E_{sc} =$	$(N_{dro} e_{dro} \delta)LCPI$	$(N_{dro} e_{dro} \delta + N_{lc} e_{lc} \delta)LCPI$
$E_{clk} =$	$(2N_{splt} e_{splt})LCPI$	$(N_{splt} e_{splt})LCPI$
$EPI =$	$E_{ac} + E_{sc} + E_{clk}$	$E_{ac} + E_{sc} + E_{clk}$

4.6.3 Energy Model

The energy metric used is energy per instruction (EPI), which is the sum of dynamic energy consumed by asynchronous cells (E_{ac}), synchronous cells (E_{sc}), and the clocking distribution network (E_{clk}) for the execution of a single instruction [4]. Expressions for E_{ac}, E_{sc}, and E_{clk} are in Table 4.4, and are functions of the following parameters: LCPI, N_{lc}, N_{dro}, e_{lc} (average energy dissipation of a logic cell), e_{dro} (energy dissipation of a DRO cell), e_{splt} (energy dissipation of a *splitter*), and δ (switching activity factor).

In conventional SFQ, no LA and FA cells are used, so $E_{ac} = 0$. However, the cost of logic gates is accounted for, along with the cost of DRO cells, in the E_{sc} expression. In xSFQ, the energy consumed by LA and FA cells is part of E_{ac}. The factor of 2 that appears in the E_{clk} expression compensates for the additional delay caused by the relax phase.

4.6.4 Energy-Delay Product Comparison

To quantify the gains of xSFQ over conventional SFQ technologies, we perform various simulations with N_H/N_I, γ, N_{lg_cp}, and o_{dp} as free variables. The assumptions are: (a) all gate inputs and outputs are buffered by JTLs, (b) the clock distribution network has zero skew and can be wave-pipelined (common practice in conventional SFQ designs), (c) xSFQ FA and LA cells are always used in pairs (the most conservative case), (d) excite and relax phases are balanced, and (e) $\eta = 1.3$, similar to Hartstein and Puzak [86]. We note that our models do not include the circuitry overhead required for pipeline interlock and flushing. Moreover, $N_{lg} = 10,000$, $N_{lg\_cp} = 150$ (based on synthesis results of a RISC-V RV32I core), and $\gamma = 0.8$, if not stated differently. The 2.8 ps and 2.3 ps setup time requirements of conventional SFQ Boolean and DRO cells are also considered.

[4]As already discussed, recent energy-efficient SFQ logics have zero static power dissipation and do not sacrifice speed or compatibility with existing fabrication processes.

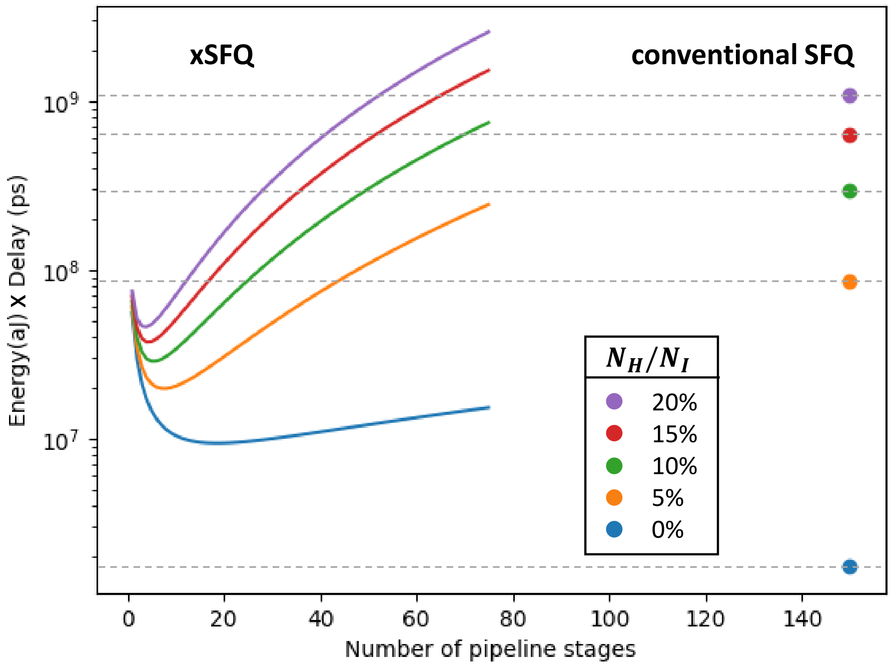


Figure 4.14: EDP versus pipeline depth comparison. $N_H/N_I \in [0,\ 0.2]$, $\gamma = 0.8$, $o_{dp} = 2.5$, $N_{lg} = 10,000$, $N_{lg\_cp} = 150$, $\eta = 1.3$. The y-axis is on a logarithmic scale.

An EDP versus pipeline depth comparison between xSFQ and conventional SFQ for various pipeline hazard rates is in Figure 4.14. In the case where $N_H = 0$, conventional SFQ achieves better results than xSFQ. This is expected, as this scenario favors very deep pipelines. LCPI is 1, each xSFQ logical cycle consumes two physical cycles, the minimum physical cycle time (t_s) is shorter for SFQ than xSFQ, and a gate-level pipelined xSFQ design requires more synchronous cells than its SFQ equivalent. However, as the number of hazards increases to more realistic values [87] and LCPI becomes greater than 1, xSFQ gains surpass conventional SFQ. Table 4.5 provides more detailed results.

In all three cases—$\gamma = 0.8, \gamma = 0.9, \gamma = 1.0$—xSFQ performs better than conventional SFQ in terms of EDP and EDDP (energy-delay^2 product) for non-zero pipeline hazard rates. More specifically, the gains increase super linearly with N_H/N_I, while the optimal

Table 4.5: EDP, EDDP, and EPI gains achieved by xSFQover conventional SFQ.

$\gamma = \mathbf{0.8}$

N_H/N_I	**EDP**	**EDDP**	**EPI**
0%	0.2x (p=19)	0.05x (p=61)	4x
5%	4.3x (p=8)	1.6x (p=13)	27x
10%	10.2x (p=5)	4.5x (p= 9)	48x
15%	16.8x (p=4)	8.1x (p=7)	67.6x
20%	23.7x (p=4)	12.2x (p=6)	86x

$\gamma = \mathbf{0.9}$

N_H/N_I	**EDP**	**EDDP**	**EPI**
0%	0.2x (p=19)	0.05x (p=61)	4x
5%	5x (p=7)	1.9x (p=13)	29.6x
10%	11.8x (p=5)	5.3x (p=9)	53x
15%	19.4x (p=4)	9.6x (p=7)	74.6x
20%	27.1x (p=3)	14.3x (p=6)	94.5x

$\gamma = \mathbf{1.0}$

N_H/N_I	**EDP**	**EDDP**	**EPI**
0%	0.2x (p=19)	0.05x (p=61)	4x
5%	5.7x (p=7)	2.2x (p=12)	32.2x
10%	13.5x (p=5)	6.2x (p=8)	58x
15%	22x (p=4)	11.1x (p=6)	81.4x
20%	30.8x (p=3)	16.5x (p=5)	103x

design point shifts to shorter pipelines. For example, for $\gamma = 0.8$ and $N_H/N_I = 5\%$, the optimum number of pipeline stages p is 8 if optimized for EDP, and 13 if optimized for EDDP. For $N_H/N_I = 10\%$ and $N_H/N_I = 20\%$, p becomes 5 and 4 for maximum EDP gains, and 9 and 6 for maximum EDDP gains. In a similar vein, increases in γ lead to higher EDP and EDDP gains and shorter pipelines. Moreover, xSFQ achieves $4\times$ lower EPI than conventional SFQ even in the case of hazard-free execution. An increase of one order of magnitude is observed for $N_H/N_I = 5\%$ and two orders of magnitude for $N_H/N_I = 20\%$.

An EPI breakdown of EDP-optimized xSFQ and conventional SFQ designs is provided in Table 4.6 ($\gamma = 0.8$). As pipeline hazards ratio increases, shorter pipelines are preferable, which leads to lower energy consumption in the synchronous elements and

Table 4.6: EPI breakdown of EDP-optimized xSFQ and conventional SFQ designs.

N_H/N_I	E_{ac} (fJ)		E_{sc} (fJ)		E_{clk} (fJ)	
	xSFQ	SFQ	xSFQ	SFQ	xSFQ	SFQ
0%	24	0	15	50	27	49
5%	32	0	7	352	11	343
10%	34	0	4	653	7	637
15%	36	0	3	954	5	931
20%	39	0	3	1,256	6	1,225

the clock distribution network (in the case of xSFQ), despite the increase in LCPI. In conventional SFQ, where pipeline depth is not configurable, the energy consumed by these components increases significantly.

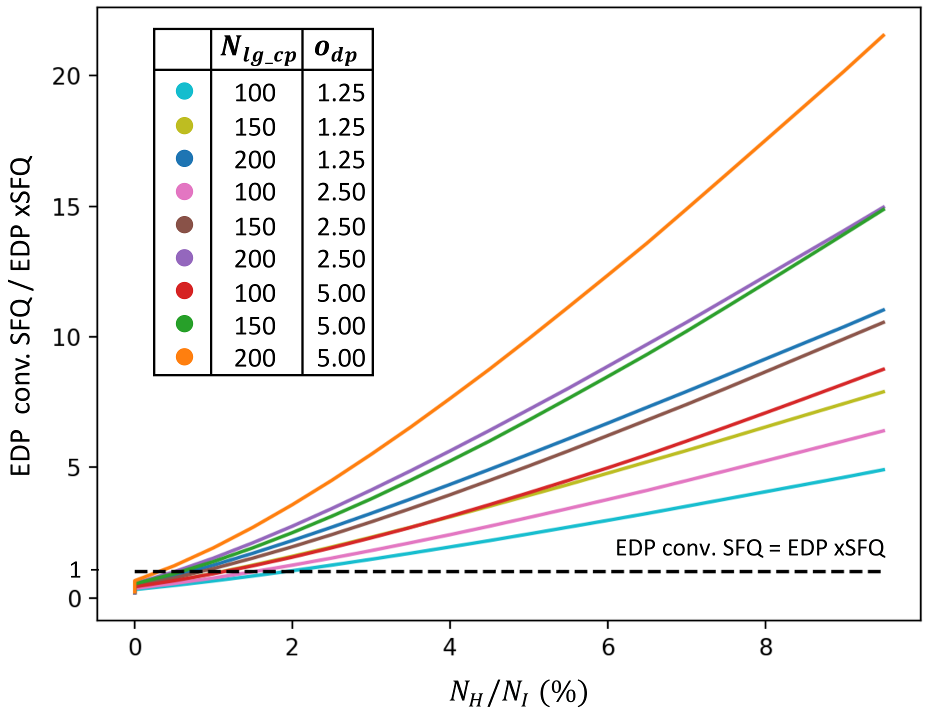


Figure 4.15: EDP conventional SFQ/xSFQ comparison for N_H/N_I ranging from 0% to 10%, N_{lg_cp} from 100 to 200, and o_{dp} from 1.25 to 5.0.

Finally, Figure 4.15 plots EDP graphs for various N_{lg_cp} and o_{dp} values ($\gamma = 0.8$). The crossover points at which xSFQ begins to demonstrate EDP improvements over conventional SFQ is below a 2% hazard rate for all cases and shifts to the left as N_{lg_cp} and o_{dp} increase. More specifically, EDP is more sensitive to the number of gates on the critical path than the overhead for the padding of uneven paths. For example, the crossover point for $N_{lg\_cp} = 100$ and $o_{dp} = 2.5$ is at $N_H/N_I = 2\%$, for $N_{lg\_cp} = 200$ and $o_{dp} = 2.5$ at $N_H/N_I = 1\%$, and for $N_{lg\_cp} = 100$ and $o_{dp} = 5.0$ at $N_H/N_I = 1.5\%$. For a 10% hazard rate, the EDP gains achieved by xSFQ over conventional SFQ for these three cases are 6.1×, 14.6×, and 8.5×, respectively, excluding the cost of interlock logic.

4.7 Conclusion

In contrast to conventional superconducting approaches that rely on a clock being delivered to each and every gate in the system, xSFQ leverages an alternating, unordered encoding for the design of completely clock-free combinational logic elements. Clocking is then used only for the synchronization of storage elements (e.g., latches), similar to CMOS, and sequential network designs have a conventional look-and-feel. To verify our hypothesis, we designed analog circuit models for all proposed logic elements and validated their operation in a superconducting-aware version of SPICE. We also built a discrete-event simulation framework [11] that aids in the evaluation of more complex systems and used it to demonstrate the effectiveness of the presented optimizations, as well as the versatility of xSFQ.

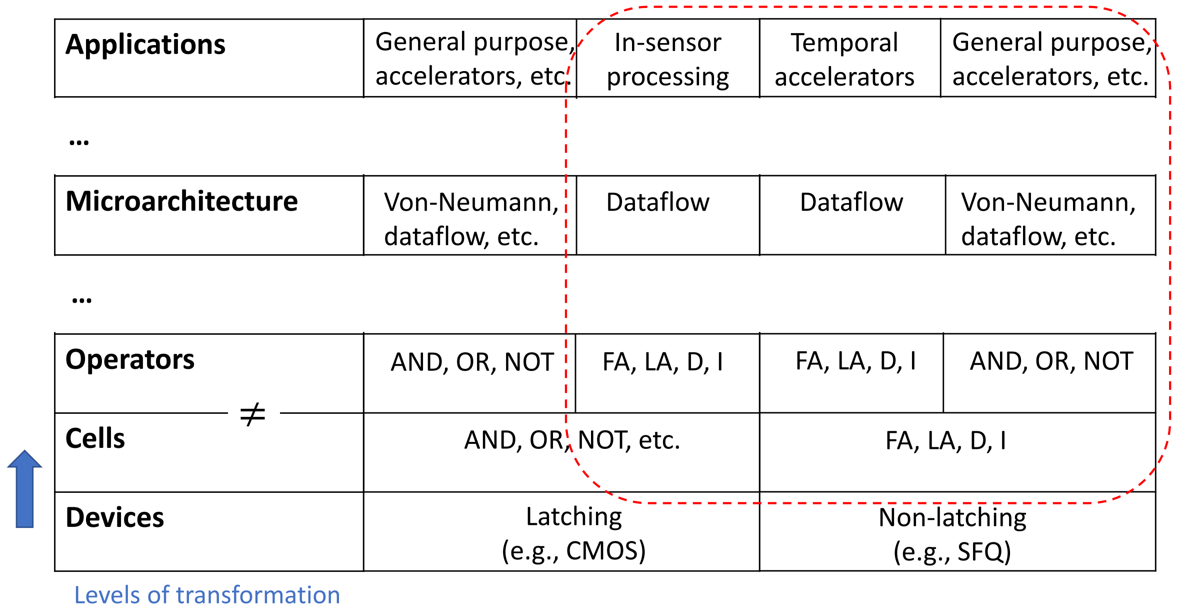
Like any new logic family, there are relative advantages and disadvantages compared to prior approaches. However, we find through a detailed analysis of energy-delay product for pipelined designs that xSFQ is superior to conventional SFQ in nearly all use-cases. Exceptions in which this is not the case are jitter-free designs with almost perfectly-balanced delay paths and hundreds of pipeline stages that have 99% stall-free operation. If even a few levels of logic are desirable between (architectural) pipeline stages, xSFQ is superior in terms of EDP, EDDP, and EPI. For example, for a design resembling a RISC-V RV32I core and a 20% pipeline hazards, xSFQ achieves 31× EDP, 17× EDDP, and 103× EPI gains compared to conventional SFQ. We expect these gains to be even more significant if the overhead of interlock logic is accounted for.

The provision of xSFQ's dual-rail construction, along with its alternating periods of excitation and relaxation, introduces new opportunities for phase rebalancing optimizations. Circuit- and gate-level enhancements that further exploit this logical framework are likely possible. Moreover, the freedom from excessively deep pipelines unlocks new architectural opportunities and makes the design process more familiar to those coming from more traditional digital design backgrounds. Finally, the presented analytical power-performance models allow for exploring the impact of pipeline depth on the efficiency of superconductor microprocessors and other non-streaming applications, as well as opening pathways for identifying the technology's key architectural challenges.

Chapter 5

Tomorrow's Outlook

A bird's eye view of the contributions discussed so far, along with the author's vision, are shown in Figure 5.1.



Applications	General purpose, accelerators, etc.	In-sensor processing	Temporal accelerators	General purpose, accelerators, etc.
...				
Microarchitecture	Von-Neumann, dataflow, etc.	Dataflow	Dataflow	Von-Neumann, dataflow, etc.
...				
Operators	AND, OR, NOT	FA, LA, D, I	FA, LA, D, I	AND, OR, NOT
≠				
Cells	AND, OR, NOT, etc.		FA, LA, D, I	
Devices	Latching (e.g., CMOS)		Non-latching (e.g., SFQ)	

Levels of transformation

Figure 5.1: Overview of this dissertation's contributions and vision (outlined in red).

By making a clear distinction between hardware cells and logical operators, it is possible to repurpose Boolean cells in level-based or latching technologies for the efficient implementation of temporal functions and to establish a temporal foundation for practical pulse-based or non-latching computing without imposing any logic restrictions. To support these arguments, we (a) demonstrated the applicability of race logic at scale and showed its promise for in-sensor AI; (b) built temporal SFQ accelerators that do not inherit the shortcomings of fully-synchronous superconductor systems; and (c) transformed the common understanding of how general-purpose computing can be implemented in superconducting logic through a new fluxon-clearing style of logic and architecture.

Looking forward, we foresee a plethora of new research opportunities at the intersection of these advancements and new classes of application—e.g., neuromorphic or quantum—and technologies—e.g., optics. However, for the rest of this chapter, our focus is on two ideas that go beyond pure information processing, are critical for the success of future endeavors, and lend a new perspective to the design of superconductor memory systems and formally verifiable hardware.

5.1 Superconductor Delay-Line Memory

Despite advances in fabrication [55], tools [88, 11], and logic levels [8, 9, 10], the lack of a reliable high-speed and high-density superconductor memory in many cases impedes the long-term development of practical superconductor systems [89]. Unfortunately, direct application of SFQ principles to memory leads to designs with low access latency but insufficient density [90]. Recently-proposed arrays of vortex transition (VT) cells open a path forward and promise up to 1 Mbit of data per square centimeter [49]–as extrapolated by the measured area, each cell takes 99 μm^2. However, surpassing this point becomes extremely challenging as the dimensions of a VT cell are defined by the length of flux transformers, which do not scale favorably. Hybrid architectures that combine SFQ and CMOS technologies provide better scalability [91, 92, 93]. Nevertheless, CMOS units are slow, usually reside outside of the 4.2 Kelvin cryocooler in which superconductor electronics are placed, and consume a significant amount of power, especially due to their resistive interconnects. A great deal of effort has also been invested in memory cells employing special magnetic materials [94, 95, 96, 97, 98]. While encouraging in many aspects, these approaches suffer from complex device structures, and thus do not scale as desired. Finally, nanowire-based memories provide an interesting alternative [99, 100, 101]. In contrast to the attempts discussed above, nanowire-based memories rely on kinetic, rather than geometric, inductance; therefore, they do not have the same miniaturization challenges. Moreover, the use of thermally-coupled cryotrons and row-select heaters eliminates the need for external addressing circuitry in nanowire memories [101]. But the thermal variation introduced by the heaters may also cause unwanted side effects, with fluctuation in the kinetic inductance of the hTron-channels being a prime example.

Here, we take an approach that departs from grid-like memory structures, and present a superconductor delay-line memory based on passive transmission lines (PTLs). PTLs are lengths of superconducting wire that convey single flux quanta without resistive attenuation and are traditionally used for energy-efficient routing in large-scale SFQ designs [102]. We claim that they can also be thought of as memory devices if used to form loops. Figure 5.2 illustrates one such loop, where SFQ pulses are introduced at one end of the line, travel along it for a given time, are picked up at the output end, and again transmitted to the input to repeat the cycle.

Reading from and writing to a delay-line memory is accomplished by interrogating locations (addresses) in a time-serial way. This serialization allows the time sharing of control circuitry and forgoes data splitting and merging, which leads to minimal hardware overhead for addressing. Therefore, the density of a delay-line memory is primarily defined by the operating speed of the interface circuitry (f), the pulse travel speed in the

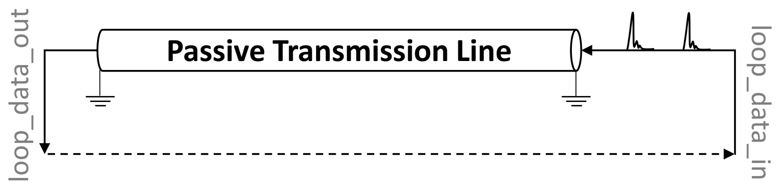


Figure 5.2: Incoming SFQ pulses (*loop_data_in*) are introduced at one end of the PTL, travel along it at a controlled speed for a given time, are picked up at the output end of the line (*loop_data_out*), and again transmitted to the input for repetition of the cycle.

delay line (v), the delay line pitch (w), and the number of vertical layers (N).

$$Density = \frac{f}{v \times w} \times N \tag{5.1}$$

Our results indicate that the SFQ travel speed in PTLs can be as low as 0.007× the speed of light, which may lead to densities in the 10s to the 100s of Mbit/cm^2. More aggressive approaches, based on vertically stacked high kinetic inductors, promise another order of magnitude improvements and invite a more in-depth level of investigation.

5.1.1 Memory Controller Microarchitecture

The configuration of the proposed passive superconducting memory system is illustrated in Figure 5.3 (i). The design consists of a superconducting delay line and a control logic block. The delay line serves as the circulating loop storage and delays any data that arrives at its input (*loop_data_in*). The delay introduced by the loop depends on the line's length and the pulse travel speed in the line. At the end of each round trip, the delayed data at the output of the line (*loop_data_out*) feeds into the controller, which serves as a memory interface. The controller is responsible for deciding whether signals from the feedback path (*loop_data_out*) or the input (*write_data*) will be forwarded to the delay line (*loop_data_in*) for another round, or copied and forwarded to the readout port (*read_data*).

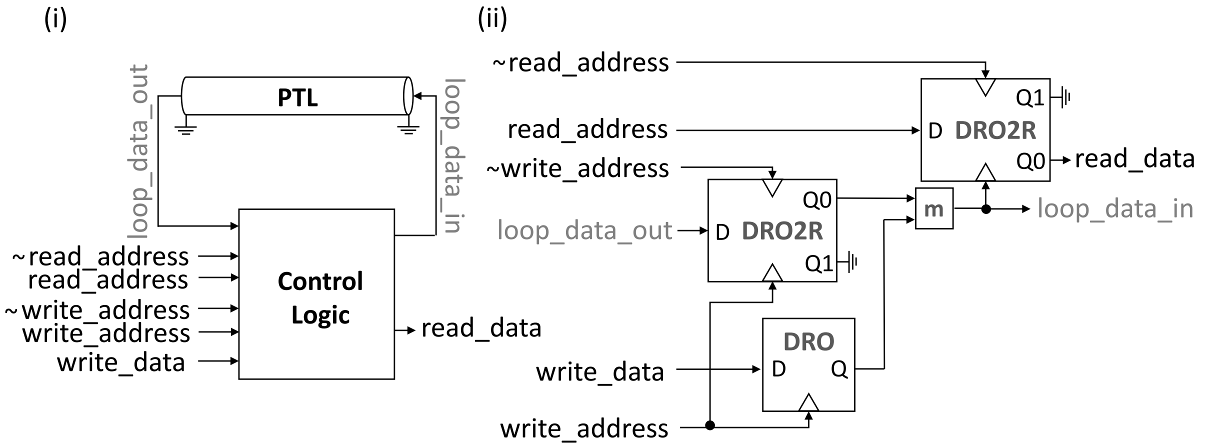


Figure 5.3: Panel (i): Overview of the proposed PTL-based superconductor delay-line memory system. Panel (ii): Block diagram of a control logic block enabling sequential-access addressing. A destructive readout (DRO) cell stores a *write_data* signal until *write_address* arrives. A DRO cell with two read-out ports (DRO2R) is used to retime and respace the data pulse coming out of the delay line when *write_address* is low (its complement, ∽*write_address*, will be high and serve as a synchronizing pulse) or filter it out otherwise. For reading, another DRO2R cell is used. In this case, a high *read_address* loads this DRO2R cell and a subsequent *loop_data_in* or ∽*read_address* pulse triggers it. If *loop_data_in* arrives first, the stored SFQ is forwarded to the *read_data* line; otherwise, it is sent to the ground. A merger cell, denoted by *m*, is used to wire two lines together: the first from the output of the DRO for a write operation, and the second from the DRO2R that continues rotating any SFQ that already exist in the loop. A splitter cell, represented by a black circle, is used for a fan-out of two.

The block diagram of the controller is provided in Figure 5.3 (ii). Temporal signals, generated by comparing the value of an address counter with a target address, are used for addressing. The Merger cell, denoted with the letter *m*, stitches together and forwards all signals that appear on its two input lines to its single output line. Thus, when the *write_addr* signal is low and no pulse appears on the corresponding line in the designated interval, its complementary signal, ∽*write_addr*, is high and the *loop_data_out* signal flows from the DRO2R (DRO with two outputs) on the left into the delay line input (*loop_data_in*) uninterrupted. Otherwise, when the *write_addr* signal is high, the *loop_data_out* signal is ignored, the content of the DRO2R is cleared, and *write_data* is forwarded to the delay line input and readout circuitry. The use of differential signalling for the *write_address* enables the correction of data timing distortions in both the control circuitry and memory loop. The readout circuitry on the right also consists of a DRO2R cell, which in this case is loaded by the *read_address* signal. As with the first DRO2R, there are two cases: in the first, the arrival of a pulse on *loop_data_in* pushes the stored value to the Q0 output port (*read_data*); in the second, the complementary $\backsim$*read_address* signal clears the cell, flushing the stored value.

Simulation results for a superconducting memory system built with this controller and a passive delay line are provided in Figure 5.4. In the shown examples, three memory addresses are supported, and the memory controller operates at 100 GHz. A high *write_data* signal is provided before interval 0 of the first rotation, or trip 0, and both the *write_address* and *read_address* lines are asserted in interval 1. Upon the arrival of the *write_address* signal, a pulse appears on the *loop_data_in* line, which demonstrates a successful memory write operation. The appearance of a *read_data* pulse after the arrival of the *read_address* signal also indicates that write operations have higher priority than read. To illustrate the non-destructive nature of readout, in the second round trip of Figure 5.4 (i), *read_address* is set to 1 again but no *write_address* is provided this time. A pulse appears on the *read_data* line, satisfying the specification. In a similar fashion, to demonstrate a successful overwrite, in the second round trip of Figure 5.4 (ii), *write_address* is set to 1 but no *write_data* pulse is given (denoting a logical False). As anticipated, no pulse appears on the *loop_data_in* line after this operation.

Note that the presented memory system allows one to search and operate on all of the memory contents while waiting for the entire circulation time to pass, thereby eliminating the need to broadcast to or continuously poll individual cells. The design's rotating nature not only circumvents classic fan-in and fan-out limitations of superconductor electronics,

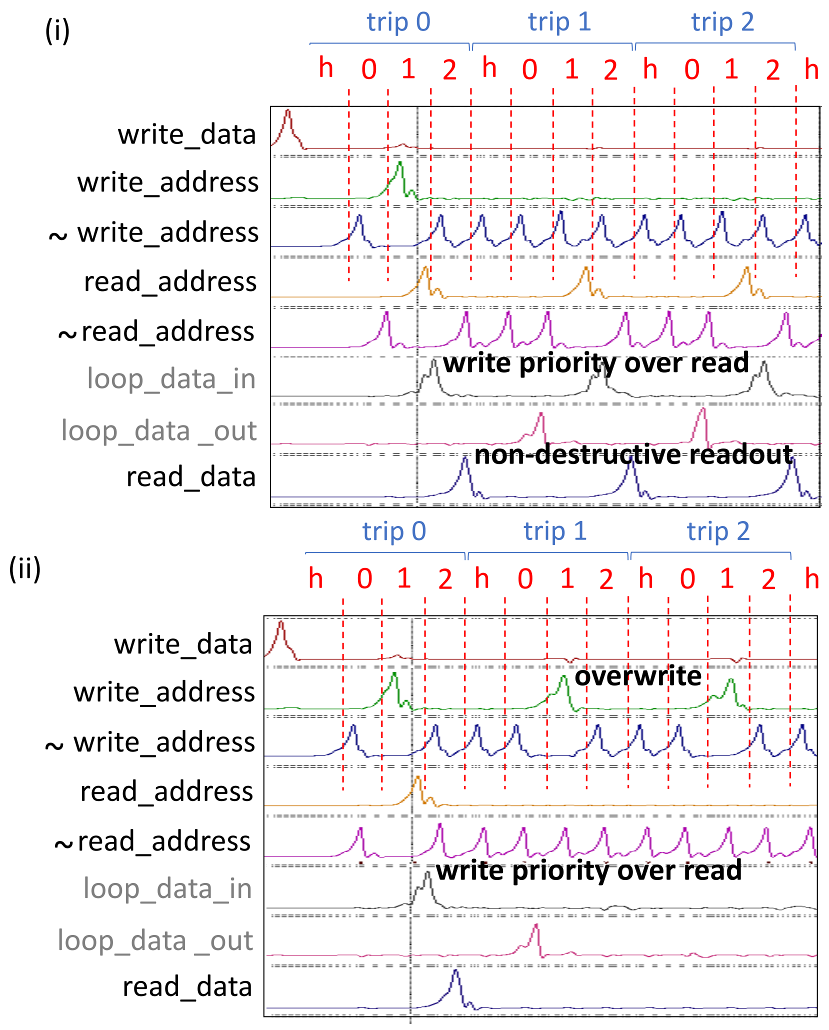


Figure 5.4: WRSPICE [54] simulation results of a passive superconducting memory system using MIT Lincoln Laboratory's SFQ5ee process parameters and operating at 100 GHz. For simplicity, the number of addresses is set to three. A *write_data* pulse must always appear before the desired *write_address*, so each round trip across the recirculating loop storage consists of four cycles, with h denoting a header interval. In both shown cases, the loop starts empty and a high *write_data* signal is provided at the beginning of trip 0. Moreover, in trip 0, both *write_address* and *read_address* are set to 1. The appearance of pulses on the *loop_data_in* and *read_data* lines verifies that write has priority over read. In trips 1 and 2 of Panel (i), *read_address* is set to 1 and no *write_address* is provided. The appearance of pulses on the *loop_data_in* and *read_data* lines indicates the non-destructive nature of readout. In trips 1 and 2 of Panel (ii), *write_address* is set to 1, but this time no *write_data* pulse is given. No pulses appear on the *loop_data_in* line, demonstrating a successful overwrite.

but also supports the addition of multiple write and read ports and the inexpensive implementation of content-addressable memories [103].

Circuit Implementation and Analysis

To evaluate the performance and feasibility of the proposed architecture, we first provide schematics and simulation results for the memory controller's main components; next, we analyze their latency; lastly, we perform a voltage bias margin analysis for the entire system, including all loading effects due to the control logic, PTL, and accompanying driver and receiver circuitry. The controller, shown in Figure 5.3 (ii), consists of a destructive readout (DRO) cell, two DRO cells with two output ports each (DRO2R), and a merger cell. The schematics and corresponding simulations for each cell are provided in Figure 5.5.

The electrical and timing properties of these cells affect both the performance and functionality of the proposed memory (or any other) system. In particular, electrical issues typically brought on by parametric variation can result in under- or over-biased Josephson junctions, which in turn can lead to their dysfunction, or delayed or early switching times. To avoid erroneous behavior and ensure correct system timing, the effects of under- and over-biasing are first examined at the cell level. Performing this bias analysis for cells in isolation, however, is not sufficient because it excludes the loading effects that are present in a system setting.

To account for loading, we iteratively measure and tune components to reach the desired timing through an *in-situ* approach—that is, while each cell is fully loaded by the remaining components in the memory controller. The results of this process are shown in Figure 5.6. Nominal delays are indicated in red. DRO and DRO2R delays are measured as the clock-to-Q delay, while merger delay is measured as the propagation delay from either input to the output. Delays in each cell increase as bias decreases, and

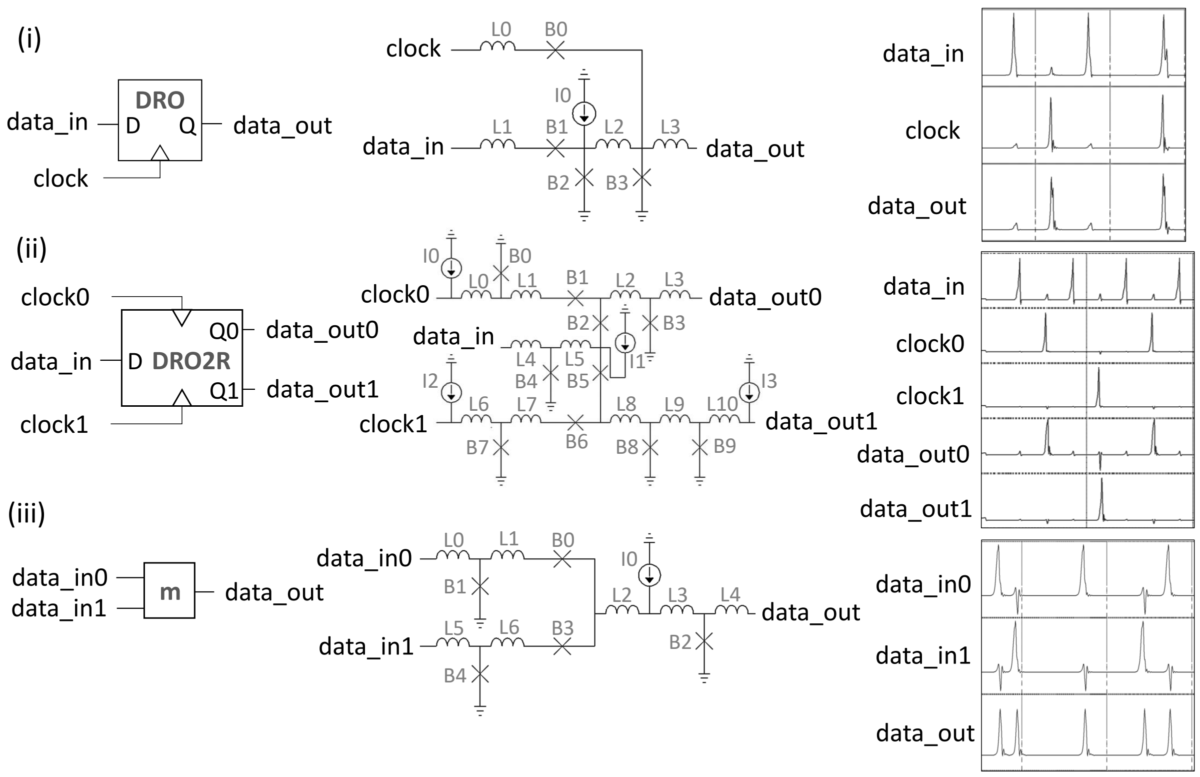


Figure 5.5: Panel (i): symbol, schematic, and simulation results of a destructive readout (DRO) cell. An incoming *data_in* SFQ pulse is stored in the superconducting quantum interference device (SQUID) formed by B2-L2-B3 until a *clock* pulse arrives. The arrival of a *clock* pulse switches B3 and releases a *data_out* SFQ pulse. Panel (ii): symbol, schematic, and simulation results of a DRO cell with two readout ports (DRO2R). The DRO2R cell performs largely the same operation as the DRO—but in this case, the storage element is shared between two parallel loops: B4-L5-B2-L2-B3 and B4-L5-B5-L8-B8. A pulse appearing on either clock input will clear the stored SFQ and push it to the respective output line. Panel (iii): symbol, schematic, and simulation results of a merger cell. This design passes incoming SFQ pulses on either of its two input ports to its output. To prevent an input SFQ from the opposite line propagating backwards to the input, two blocking Josephson junctions, B0 and B3, are used.

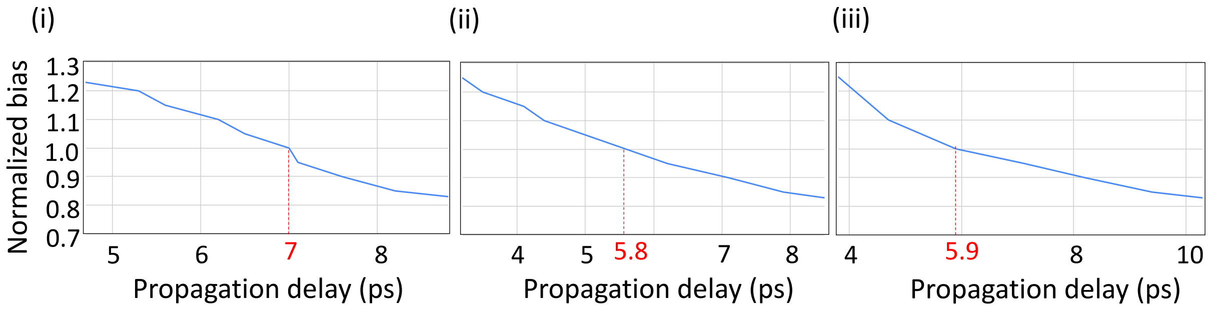


Figure 5.6: Bias versus propagation delay for DRO, DRO2R, and merger cells. Red markings indicate nominal values. To make bias margins symmetric, we center the nominal delay of each cell between its upper and lower time bounds.

decrease as bias increases. To make bias margins symmetric, we center the nominal delay of each cell between its upper and lower time bounds.

Using interval analysis and the above delays, we estimate the controller's maximum operating frequency, and repeat cell tuning and bias margin measurements. In this case, though, bias margins are measured for the overall system and not each cell. Figure 5.7 illustrates our results for frequencies ranging from 20 GHz to 100 GHz. We notice that electrical issues drive limitations in bias margin width at lower frequencies, while timing issues are the limiter at higher frequencies. This happens because timing constraints and tolerances get tighter as the address timing interval is reduced. For example, at 100 GHz, the address timing interval is just 10 ps, which leaves little room for the same variations in propagation delay that we observed in Figure 5.6. Our SPICE simulations show bias margins ranging from ±24% (at 20 GHz) to ±13% (at 100 GHz), which are well above the widely accepted ±10% threshold [104].

Data Density Estimation

The physical storage density—that is, bits per area—of a delay-line memory depends on 1) the linewidth and line spacing, set by the the fabrication process; 2) the travel speed, set by the material of choice and the line topology; 3) the relative timing between

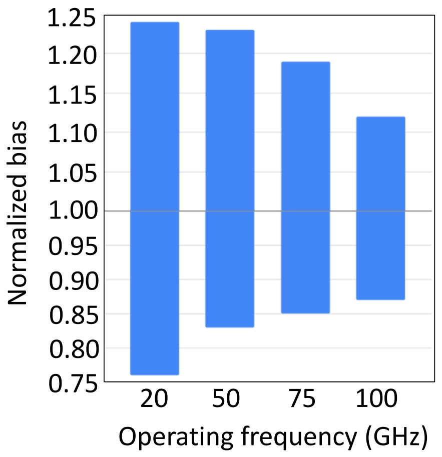


Figure 5.7: Bias margins of the proposed memory system for a variety of operating frequencies.

two adjacent bits, set by the controller's operating frequency; and 4) the number of PTL memory routing layers. We estimate the density of the proposed PTL-based superconducting delay line memory by choosing various settings for each of these free variables and summarize our results in Table 5.1.

A typical Nb stripline of 250 nm linewidth has a minimum spacing of 250 nm [55] and propagates SFQ pulses at a speed of $0.3c$. This leads to data densities of up to 0.9 Mbit/cm^2 at 100 GHz, if all four metal routing layers are used, and matches the state of the art [49]. Reducing the Nb stripline linewidth and minimum spacing from 250 nm to 120 nm is a possible but more aggressive design choice [105] and results in densities of up to 1.9 Mbit/cm^2, exceeding the state of the art by almost 2×.

By switching device material and topology to a MoN kinetic inductor microstrip with the same dimensions, available on just one layer within MIT Lincoln Laboratory's SFQ5ee [47] or SC2 [105] processes, the travel speed of pulses in the line falls by about 6×. This slowdown yields densities of up to 1.4 and 4.0 Mbit/cm^2 for 250 nm and 120 nm linewidths, respectively, at 100 GHz. Furthermore, scaling the number of layers on which the MoN kinetic inductor is available from one to four—the line topology, in this case,

Device	Linewidth (nm)	Spacing (nm)	Fabrication Process	Memory Layers	Process Maturity	Capacitance (fF/μm)	Inductance (pH/μm)	Travel Speed ($\times$c)	Frequency (GHz)	Density (Mbit/cm^2)
Nb Stripline	250	250	SFQ5ee	4	Mature	0.25	0.50	0.298	20	0.2
									50	0.4
									75	0.7
									100	0.9
	120	120	SC2	4	Aggressive	0.19	0.65	0.296	20	0.4
									50	0.9
									75	1.4
									100	1.9
MoN Kinetic Inductor Microstrip	250	250	SFQ5ee	1	Mature	0.16	32	0.047	20	0.3
									50	0.7
									75	1.1
									100	1.4
	120	120	SC2	1	Aggressive	0.14	66.70	0.034	20	0.8
									50	2.0
									75	3.0
									100	4.0
MoN Kinetic Inductor Stripline	120	120	SC2	4	Aggressive	0.19	66.70	0.029	20	3.2
									50	8.1
									75	12.1
									100	19.0
NbTiN Kinetic Inductor Stripline	100	120	Not Established	4	Academic	0.17	490.5	0.011	20	10.7
									50	26.6
									75	40.0
									100	53.3
NbN Kinetic Inductor Nanowire	40	120	Not Established	4	Academic	0.04	2,050	0.011	20	15.1
									50	37.7
									75	56.6
									100	75.4
	15	120	Not Established	4	Academic	0.04	5,467	0.007	20	28.1
									50	70.1
									75	105.2
									100	140.3
	15	120	Not Established	100	Academic	0.04	5,467	0.007	20	701.4
									50	1,753
									75	2,630
									100	3,507

Table 5.1: Memory density estimates for a variety of mature, aggressive, and academic configurations. A mature process is considered one that is well-documented and available as a fabrication option. For example, the MIT Lincoln Laboratory SFQ5ee process has served as the de-facto standard for fabrication since its introduction in 2016 [47]. In a similar fashion, a process is aggressive if it has been evaluated experimentally in real circuits, like the MIT Lincoln Laboratory SC2 [105], but is outside common design rules. The term "academic" is used to indicate that the device has been only theoretically evaluated or experimentally evaluated but not scaled. Capacitance and inductance are also provided because they contribute directly to the SFQ travel speed.

transforms into that of a stripline—increases the data density to 3.2 Mbit/cm^2 at 20 GHz and 19 Mbit/cm^2 at 100 GHz for 120 nm linewidth and 120 nm spacing.

At this point, it is evident that the use of materials with increasingly high kinetic inductance is conducive to higher densities. To this end, we explore the potential of NbTiN striplines that exhibit roughly an order of magnitude higher inductance than their MoN counterparts and propagate SFQ pulses at a speed of $0.011c$ [106]. Our results indicate that for a NbTiN stripline with 100 nm width, 120 nm spacing, four metal routing layers, and controller frequencies between 20 and 100 GHz, the estimated data densities range from 10.7 to 53.3 Mbit/cm^2.

A more forward-looking approach comes from the use of NbN kinetic inductor nanowires. In the case of an experimentally-tested NbN nanowire with 40 nm linewidth, the inductance scales to 2,050 pH/μm [107]. A roughly proportional drop in capacitance keeps the pulse travel speed the same, $0.011c$, but the reduced linewidth pushes the maximum data density to 75.4 Mbit/cm^2 at 100 GHz. A further simulated linewidth reduction to 15 nm causes the inductance to skyrocket to 5,467 pH/μm [107], which drops the travel speed to $0.007c$ and increases data density to 140.3 Mbit/cm^2 at 100 GHz. This is equivalent to a pulse spacing of 210 μm along a NbN nanowire, about 60$\times$ shorter than that in the YBCO line [108]. Moreover, if CMOS scaling techniques like very large stack-ups—such as the 100-layer stacks used in 3-D NANDs [109]—were to be adopted in the future by superconducting technology, this NbN nanowire technology could provide a memory density of 3,507 Mbit/cm^2 at 100 GHz operating speed.

5.2 Generalized Temporal Formalism

Given the long history of temporal logic for verification purposes, a question that comes up is whether a mapping between the discussed FA, LA, D, and I operators and those of linear temporal logic (LTL) is possible. This quest is partially driven by the impact of prior temporal logic and language abstractions, which allowed both systems and their properties to be represented by formulas in the same logic, in the reactive/concurrent computing field [110]. Our overarching goal here is to establish the foundation that allows the reasoning about temporal hardware, ideally through a series of logical implications.

5.2.1 LTL and Past LTL Semantics

The basic LTL operators are: $\Diamond$ *some time in the future*, $\Box$ *always in the future*, $\bigcirc$ *next time* (*tomorrow*), and $\mathbf{U}$ *until*. Past LTL (PLTL) extends LTL with past-time operators, which are the temporal duals of LTL's future-time operators, and allows one to concisely express statements on the past time instances, such as: $\blacklozenge$ *some time in the past*, $\blacksquare$ *always in the past* (*historically*), $\bullet$ *previous time* (*yesterday*), and $\mathbf{S}$ *since* [111, 112].

A definition of the semantics of PLTL is provided in Table 5.2. 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 $\langle S, t\rangle \models \phi$.

Table 5.2: Semantics of PLTL.

$\langle S,t\rangle$	$\models$	$\blacklozenge\phi$	iff	$\exists k\colon k \in [0,t].\ \langle S,k\rangle \models \phi$
$\langle S,t\rangle$	$\models$	$\blacksquare\phi$	iff	$\forall k\colon k \in [0,t].\ \langle S,k\rangle \models \phi$
$\langle S,t\rangle$	$\models$	$\bullet\phi$	iff	$t > 0$ and $\langle S,t-1\rangle \models \phi$
$\langle S,t\rangle$	$\models$	$\phi\mathbf{S}\psi$	iff	$\exists k\colon k \in [0,t].\ (\langle S,k\rangle \models \psi$ and $\forall j\colon j \in (k,t].\ \langle S,j\rangle \models \phi)$

These operators allow for the efficient representation and reasoning of propositions qualified in terms of time; e.g., an event in a system S has happened some time in the past. However, to the best of our knowledge, they have not been used to specify computation directly. Moreover, due to their propositional nature, they are incapable of encapsulating when a formula holds, which is essential for race logic. To address these issues, we first propose a mapping between PLTL and race logic operators and then introduce the earliest-occurrence function $\mathcal{E}_{\langle S,t\rangle}()$ that returns the first time step in which a PLTL formula gets satisfied.

5.2.2 Race Logic Semantics

The semantics of FA, LA, D, and I operations can be formally defined using existing PLTL operators, as shown in Table 5.3—$\CIRCLE^c$ denotes the application of $\CIRCLE$ operator c times.

Table 5.3: PLTL-based semantics of race logic operations.

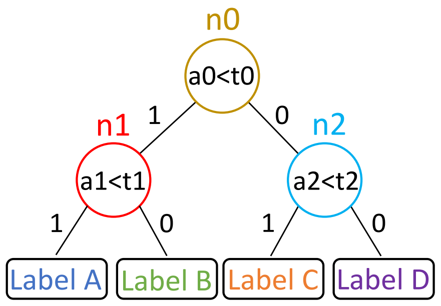
$\langle S,t\rangle$	$\models$	$\mathbf{FA}\phi\psi$	iff	$\langle S,t\rangle \models \blacklozenge\phi \vee \blacklozenge\psi$
$\langle S,t\rangle$	$\models$	$\mathbf{LA}\phi\psi$	iff	$\langle S,t\rangle \models \blacklozenge\phi \wedge \blacklozenge\psi$
$\langle S,t\rangle$	$\models$	$\mathbf{D}c\phi$	iff	$\langle S,t\rangle \models \CIRCLE^c\blacklozenge\phi$
$\langle S,t\rangle$	$\models$	$\psi\mathbf{I}\phi$	iff	$\langle S,t\rangle \models \blacklozenge(\phi \wedge \blacksquare\neg\psi)$

To extract the step at which these formulas evaluate to True for the first time, $\mathcal{E}_{\langle S,t\rangle}()$ can be used. The definition of $\mathcal{E}_{\langle S,t\rangle}()$ for a formula ϕ is given in Equation 5.2.

$$\mathcal{E}_{\langle S,t\rangle}\ (\phi) = \begin{cases} t_{min} & \exists\ t_{min}\colon\ t_{min} \in [\![\phi]\!]_{\langle S,t\rangle}.\ \langle S, t_{min}\rangle \models \phi \\ & \text{and } \forall j\colon\ j \in [0, t_{min}).\ \langle S, j\rangle \not\models \phi \\ \infty & \text{otherwise} \end{cases} \tag{5.2}$$

This earliest-occurrence function 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 $[\![\phi]\!]_{\langle S,t\rangle}$, the earliest-occurrence function returns ∞, which represents an unreachable time step.

Figure 5.8 illustrates the application of this generalized temporal formalism through our example race tree and shows how race logic designs can be fully described with PLTL operators. We believe that this description opens new questions about the verifiability advantage of temporal hardware and invites further investigation.



Label A $= \blacklozenge(a0 \wedge \blacksquare\neg t0) \wedge \blacklozenge(a1 \wedge \blacksquare\neg t1)$

Label B $= \blacklozenge(a0 \wedge \blacksquare\neg t0) \wedge \blacklozenge(t1 \wedge \bullet\blacksquare\neg a1)$

Label C $= \blacklozenge(t0 \wedge \bullet\blacksquare\neg a0) \wedge \blacklozenge(a2 \wedge \blacksquare\neg t2)$

Label D $= \blacklozenge(t0 \wedge \bullet\blacksquare\neg a0) \wedge \blacklozenge(t2 \wedge \bullet\blacksquare\neg a2)$

Figure 5.8: PLTL-based description of an example race tree design.

Bibliography

[1] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, *There's plenty of room at the top: What will drive computer performance after moore's law?*, *Science* **368** (2020), no. 6495 eaam9744.

[2] L. Ceze, M. D. Hill, and T. F. Wenisch, *Arch2030: A vision of computer architecture research over the next 15 years*, 2016.

[3] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[4] E. A. Lee, *Computing needs time*, *Commun. ACM* **52** (may, 2009) 70–79.

[5] G. Tzimpragos, A. Madhavan, D. Vasudevan, D. Strukov, and T. Sherwood, *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, (New York, NY, USA), p. 215–228, Association for Computing Machinery, 2019.

[6] G. Tzimpragos, A. Madhavan, D. Vasudevan, D. Strukov, and T. Sherwood, *In-sensor classification with boosted race trees*, *Commun. ACM* **64** (may, 2021) 99–105.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, *Journal of Machine Learning Research* **12** (2011) 2825–2830.

[8] G. Tzimpragos, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Volk, J. Shalf, and T. Sherwood, *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, (New York, NY, USA), p. 435–448, Association for Computing Machinery, 2020.

[9] G. Tzimpragos, J. E. Volk, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Shalf, and T. Sherwood, *Temporal computing with superconductors*, *IEEE Micro* (2021) 1–1.

[10] G. Tzimpragos, J. Volk, A. Wynn, J. E. Smith, and T. Sherwood, *Superconducting computing with alternating logic elements*, in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 651–664, 2021.

[11] M. Christensen, G. Tzimpragos, H. Kringen, J. Volk, T. Sherwood, and B. Hardekopf, *PyLSE: A pulse-transfer level language for superconductor electronics*, in *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2022, (New York, NY, USA), Association for Computing Machinery, 2022.

[12] G. Tzimpragos, J. Volk, A. Wynn, E. Golden, and T. Sherwood, *Pulsar: A superconducting delay-line memory*, 2022.

[13] A. Pnueli, *The temporal logic of programs*, in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57, ieee, 1977.

[14] A. Madhavan, T. Sherwood, and D. Strukov, *Race logic: A hardware acceleration for dynamic programming algorithms*, *ACM SIGARCH Computer Architecture News* **42** (2014), no. 3 517–528.

[15] J. E. Smith, *Space-time algebra: A model for neocortical computation*, in *Proceedings of the International Symposium of Computer Architecture*, ISCA '18, 2018.

[16] A. Madhavan, T. Sherwood, and D. Strukov, *Energy efficient computation with asynchronous races*, in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2016.

[17] A. Bermak and D. Martinez, *A compact 3d vlsi classifier using bagging threshold network ensembles*, *IEEE Transactions on Neural Networks* **14** (Sept, 2003) 1097–1109.

[18] X. Guo, X. Qi, and J. G. Harris, *A time-to-first-spike cmos image sensor*, *IEEE Sensors Journal* **7** (2007), no. 8 1165–1175.

[19] P. Lichtsteiner, C. Posch, and T. Delbruck, *A 128×128 120db 15μs latency asynchronous temporal contrast vision sensor*, *IEEE journal of solid-state circuits* **43** (2008), no. 2 566–576.

[20] C. Posch, D. Matolin, R. Wohlgenannt, M. Hofstätter, P. Schön, M. Litzenberger, D. Bauer, and H. Garn, *Biomimetic frame-free hdr camera with event-driven pwm image/video sensor and full-custom address-event processor*, in *Biomedical*

Circuits and Systems Conference (BioCAS), 2010 IEEE, pp. 254–257, IEEE, 2010.

[21] X. Qi, X. Guo, and J. G. Harris, *A time-to-first spike cmos imager*, in *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, vol. 4, pp. IV–824, IEEE, 2004.

[22] C. Niclass, M. Soga, H. Matsubara, S. Kato, and M. Kagami, *A 100-m range 10-frame/s 340×96-pixel time-of-flight depth sensor in 0.18-μm cmos*, *IEEE Journal of Solid-State Circuits* **48** (2013), no. 2 559–572.

[23] V. Chan, S.-C. Liu, and A. van Schaik, *Aer ear: A matched silicon cochlea pair with address event representation interface*, *IEEE Transactions on Circuits and Systems I: Regular Papers* **54** (2007), no. 1 48–59.

[24] M. H. Najafi, D. J. Lilja, M. Riedel, and K. Bazargan, *Power and area efficient sorting networks using unary processing*, in *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 125–128, Nov, 2017.

[25] J. Clow, G. Tzimpragos, D. Dangwal, S. Guo, J. McMahan, and T. Sherwood, *A pythonic approach for rapid hardware prototyping and instrumentation*, in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–7, Sept, 2017.

[26] C. Wolf and J. Glaser, *Yosys–a free verilog synthesis suite*, in *Submitted to: Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip), Linz, Austria*, vol. 10, 2013.

[27] D. Vasudevan, A. Butko, G. Michelogiannakis, D. Donofrio, and J. Shalf, *Towards an integrated strategy to preserve digital computing performance scaling using emerging technologies*, in *High Performance Computing* (J. M. Kunkel, R. Yokota, M. Taufer, and J. Shalf, eds.), (Cham), pp. 115–123, Springer International Publishing, 2017.

[28] S. Chen, Y. Wang, X. Lin, Q. Xie, and M. Pedram, *Performance prediction for multiple-threshold 7nm-finfet-based circuits operating in multiple voltage regimes using a cross-layer simulation framework*, in *2014 SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pp. 1–2, Oct, 2014.

[29] J. Shalf, S. Dosanjh, and J. Morrison, *Exascale computing technology challenges*, in *High Performance Computing for Computational Science – VECPAR 2010* (J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, eds.), (Berlin, Heidelberg), pp. 1–25, Springer Berlin Heidelberg, 2011.

[30] S. Satpathy, K. Sewell, T. Manville, Y.-P. Chen, R. Dreslinski, D. Sylvester, T. Mudge, and D. Blaauw, *A 4.5tb/s 3.4tb/s/w 64×64 switch fabric with self-updating least-recently-granted priority and quality-of-service arbitration in 45nm cmos*, in *2012 IEEE International Solid-State Circuits Conference*, pp. 478–480, 2012.

[31] S. Borkar, *Future of interconnect fabric: A contrarian view*, in *Proceedings of the 12th ACM/IEEE International Workshop on System Level Interconnect Prediction*, SLIP '10, (New York, NY, USA), pp. 1–2, ACM, 2010.

[32] J. H. Friedman, *Greedy function approximation: A gradient boosting machine*, *Annals of Statistics* **29** (2000) 1189–1232.

[33] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G.-Y. Wei, *A 28nm SoC with a 1.2 GHz 568nJ/prediction sparse deep-neural-network engine with ¿ 0.1 timing error rate tolerance for IoT applications*, in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pp. 242–243, IEEE, 2017.

[34] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, *A 640m pixel/s 3.65 mw sparse event-driven neuromorphic object recognition processor with on-chip learning*, in *VLSI Circuits (VLSI Circuits), 2015 Symposium on*, pp. C50–C51, IEEE, 2015.

[35] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, *Minerva: Enabling low-power, highly-accurate deep neural network accelerators*, in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 267–278, IEEE Press, 2016.

[36] J. Kung, D. Kim, and S. Mukhopadhyay, *A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses*, in *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pp. 85–90, IEEE, 2015.

[37] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, *Backpropagation for energy-efficient neuromorphic computing*, in *Advances in Neural Information Processing Systems*, pp. 1117–1125, 2015.

[38] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, *Sparse coding via thresholding and local competition in neural circuits*, *Neural Computation* **20** (Oct, 2008) 2526–2563.

[39] S. Naraghi, *Time-based analog to digital converters.* PhD thesis, University of Michigan, 2009.

[40] M. Bavandpour, M. R. Mahmoodi, and D. B. Strukov, *Energy-efficient time-domain vector-by-matrix multiplier for neurocomputing and beyond*, *IEEE Transactions on Circuits and Systems II: Express Briefs* (2019) 1–1.

[41] D. S. Holmes, A. M. Kadin, and M. W. Johnson, *Superconducting computing in large-scale hybrid systems*, *Computer* **48** (Dec, 2015) 34–42.

[42] D. E. Kirichenko, S. Sarwana, and A. F. Kirichenko, *Zero static power dissipation biasing of rsfq circuits*, *IEEE Transactions on Applied Superconductivity* **21** (2011), no. 3 776–779.

[43] K. K. Likharev and V. K. Semenov, *Rsfq logic/memory family: a new josephson-junction technology for sub-terahertz-clock-frequency digital systems*, *IEEE Transactions on Applied Superconductivity* **1** (March, 1991) 3–28.

[44] V. Semenov, E. Golden, and S. K. Tolpygo, *Sfq bias for sfq digital circuits*, *IEEE Transactions on Applied Superconductivity* (2021) 1–1.

[45] I. I. Soloviev, N. V. Klenov, S. V. Bakurskiy, M. Y. Kupriyanov, A. L. Gudkov, and A. S. Sidorenko, *Beyond moore's technologies: operation principles of a superconductor alternative*, *Beilstein Journal of Nanotechnology* **8** (Dec, 2017) 2689–2710.

[46] A. Holmes, M. R. Jokar, G. Pasandi, Y. Ding, M. Pedram, and F. T. Chong, *Nisq+: Boosting quantum computing power by approximating quantum error correction*, in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 556–569, 2020.

[47] S. K. Tolpygo, V. Bolkhovsky, T. J. Weir, A. Wynn, D. E. Oates, L. M. Johnson, and M. A. Gouker, *Advanced fabrication processes for superconducting very large-scale integrated circuits*, *IEEE Transactions on Applied Superconductivity* **26** (2016), no. 3 1–10.

[48] F. Ware, L. Gopalakrishnan, E. Linstadt, S. A. McKee, T. Vogelsang, K. L. Wright, C. Hampel, and G. Bronner, *Do superconducting processors really need cryogenic memories? the case for cold dram*, in *Proceedings of the International Symposium on Memory Systems*, MEMSYS '17, (New York, NY, USA), p. 183–188, Association for Computing Machinery, 2017.

[49] V. K. Semenov, Y. A. Polyakov, and S. K. Tolpygo, *Very large scale integration of josephson-junction-based superconductor random access memories*, *IEEE Transactions on Applied Superconductivity* **29** (2019), no. 5 1–9.

[50] S. K. Tolpygo, *Superconductor digital electronics: Scalability and energy efficiency issues*, *Low Temperature Physics* **42** (2016), no. 5 361–379.

[51] S. V. Rylov, *Clockless dynamic sfq and gate with high input skew tolerance*, *IEEE Transactions on Applied Superconductivity* **29** (2019), no. 5 1–5.

[52] Z. J. Deng, N. Yoshikawa, S. R. Whiteley, and T. Van Duzer, *Self-timing and vector processing in rsfq digital circuit technology*, *IEEE Transactions on Applied Superconductivity* **9** (1999), no. 1 7–17.

[53] H. R. Gerber, C. J. Fourie, and W. J. Perold, *Rsfq-asynchronous timing (rsfq-at): a new design methodology for implementation in cad automation*, *IEEE Transactions on Applied Superconductivity* **15** (2005), no. 2 272–275.

[54] W. R. Incorporated, *WRspice reference manual*, tech. rep., June, 2019.

[55] S. K. Tolpygo, V. Bolkhovsky, R. Rastogi, S. Zarr, A. L. Day, T. J. Weir, A. Wynn, and L. M. Johnson, *Developments toward a 250-nm, fully planarized fabrication process with ten superconducting layers and self-shunted josephson junctions*, *2017 16th International Superconductive Electronics Conference (ISEC)* (Jun, 2017).

[56] D. A. Reynolds and G. Metze, *Fault detection capabilities of alternating logic*, *IEEE Transactions on Computers* (1978), no. 12 1093–1098.

[57] B. Bose, *On unordered codes*, *IEEE Transactions on Computers* (1991), no. 2 125–131.

[58] J. E. Smith, *On separable unordered codes*, *IEEE transactions on computers* **100** (1984), no. 8 741–743.

[59] I. Kurosawa, H. Nakagawa, M. Aoyagi, M. Maezawa, Y. Kameda, and T. Nanya, *A basic circuit for asynchronous superconductive logic using rsfq gates*, *Superconductor Science and Technology* **9** (1996).

[60] M. Maezawa, I. Kurosawa, M. Aoyagi, H. Nakagawa, Y. Kameda, and T. Nanya, *Rapid single-flux-quantum dual-rail logic for asynchronous circuits*, *IEEE Transactions on Applied Superconductivity* **7** (1997), no. 2 2705–2708.

[61] C. E. Leiserson and J. B. Saxe, *Retiming synchronous circuitry*, *Algorithmica* **6** (June, 1991) 5–35.

[62] K. Asanović and D. A. Patterson, *Instruction sets should be free: The case for risc-v*, *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146* (2014).

[63] G. Tang, K. Takata, M. Tanaka, A. Fujimaki, K. Takagi, and N. Takagi, *4-bit bit-slice arithmetic logic unit for 32-bit rsfq microprocessors*, *IEEE Transactions on Applied Superconductivity* **26** (Jan, 2016) 1–6.

[64] G. Tang, P. Qu, X. Ye, and D. Fan, *Logic design of a 16-bit bit-slice arithmetic logic unit for 32-/64-bit rsfq microprocessors*, *IEEE Transactions on Applied Superconductivity* **28** (June, 2018) 1–5.

[65] Y. Yamanashi, M. Tanaka, A. Akimoto, H. Park, Y. Kamiya, N. Irie, N. Yoshikawa, A. Fujimaki, H. Terai, and Y. Hashimoto, *Design and implementation of a pipelined bit-serial sfq microprocessor,* core1β, *IEEE Transactions on Applied Superconductivity* **17** (June, 2007) 474–477.

[66] M. Dorojevets, P. Bunyk, and D. Zinoviev, *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** (March, 2001) 326–332.

[67] Y. Ando, R. Sato, M. Tanaka, K. Takagi, N. Takagi, and A. Fujimaki, *Design and demonstration of an 8-bit bit-serial rsfq microprocessor: Core e4*, *IEEE Transactions on Applied Superconductivity* **26** (Aug, 2016) 1–5.

[68] C. L. Ayala, T. Tanaka, R. Saito, M. Nozoe, N. Takeuchi, and N. Yoshikawa, *Mana: A monolithic adiabatic integration architecture microprocessor using 1.4-zj/op unshunted superconductor josephson junction devices*, *IEEE Journal of Solid-State Circuits* **56** (2021), no. 4 1152–1165.

[69] M. A. Green, *the cost of coolers for cooling superconducting devices at temperatures at 4.2 k, 20 k, 40 k and 77 k*, .

[70] S. S. Tannu, P. Das, M. L. Lewis, R. Krick, D. M. Carmean, and M. K. Qureshi, *A case for superconducting accelerators*, in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, CF '19, (New York, NY, USA), p. 67–75, Association for Computing Machinery, 2019.

[71] G. M. Lilly, *Device for and method of one-way cryptographic hashing*, Dec. 7, 2004. US Patent 6,829,355.

[72] K. Ishida, I. Byun, I. Nagaoka, K. Fukumitsu, M. Tanaka, S. Kawakami, T. Tanimoto, T. Ono, J. Kim, and K. Inoue, *Supernpu: An extremely fast neural processing unit using superconducting logic devices*, in *Proceedings. 53th Annual IEEE/ACM International Symposium on Microarchitecture, 2020. MICRO-53.*, 2020.

[73] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu,

K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, *In-datacenter performance analysis of a tensor processing unit*, in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, (New York, NY, USA), p. 1–12, Association for Computing Machinery, 2017.

[74] R. Cai, A. Ren, O. Chen, N. Liu, C. Ding, X. Qian, J. Han, W. Luo, N. Yoshikawa, and Y. Wang, *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, (New York, NY, USA), p. 567–578, Association for Computing Machinery, 2019.

[75] M. C. Hansen, H. Yalcin, and J. P. Hayes, *Unveiling the iscas-85 benchmarks: a case study in reverse engineering*, *IEEE Design Test of Computers* **16** (1999), no. 3 72–80.

[76] G. Pasandi, A. Shafaei, and M. Pedram, *Sfqmap: A technology mapping tool for single flux quantum logic circuits*, in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2018.

[77] K. Ishida, M. Tanaka, I. Nagaoka, T. Ono, S. Kawakami, T. Tanimoto, A. Fujimaki, and K. Inoue, *32 ghz 6.5 mw gate-level-pipelined 4-bit processor using superconductor single-flux-quantum logic*, in *2020 IEEE Symposium on VLSI Circuits*, pp. 1–2, 2020.

[78] J. M. Berger, *A note on error detection codes for asymmetric channels*, *Information and control* **4** (1961), no. 1 68–73.

[79] G. R. Trimble, *The ibm 650 magnetic drum calculator*, *IEEE Annals of the History of Computing* **8** (jan, 1986) 20–29.

[80] D. S.-M. Ho, *The study of a totally self-checking adder.*, tech. rep., ILLINOIS UNIV URBANA COORDINATED SCIENCE LAB, 1972.

[81] I. Nagaoka, M. Tanaka, K. Sano, T. Yamashita, A. Fujimaki, and K. Inoue, *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)*, pp. 1–3, 2019.

[82] S. R. Kunkel and J. E. Smith, *Optimal pipelining in supercomputers*, *ACM SIGARCH Computer Architecture News* **14** (1986), no. 2 404–411.

[83] P. K. Dubey and M. J. Flynn, *Optimal pipelining*, *Journal of Parallel and Distributed Computing* **8** (1990), no. 1 10–19.

[84] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, and P. Shivakumar, *The optimal logic depth per pipeline stage is 6 to 8 fo4 inverter delays*, in *Proceedings 29th Annual International Symposium on Computer Architecture*, pp. 14–24, 2002.

[85] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, and P. G. Emma, *Optimizing pipelines for power and performance*, in *35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002. (MICRO-35). Proceedings.*, pp. 333–344, 2002.

[86] A. Hartstein and T. R. Puzak, *Optimum power/performance pipeline depth*, in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pp. 117–125, IEEE, 2003.

[87] A. Limaye and T. Adegbija, *A workload characterization of the spec cpu2017 benchmark suite*, in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 149–158, 2018.

[88] C. J. Fourie, K. Jackman, M. M. Botha, S. Razmkhah, P. Febvre, C. L. Ayala, Q. Xu, N. Yoshikawa, E. Patrick, M. Law, Y. Wang, M. Annavaram, P. Beerel, S. Gupta, S. Nazarian, and M. Pedram, *ColdFlux superconducting EDA and TCAD tools project: Overview and progress*, *IEEE Transactions on Applied Superconductivity* **29** (2019), no. 5 1–7.

[89] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, *Energy-efficient superconducting computing—power budgets and requirements*, *IEEE Transactions on Applied Superconductivity* **23** (2013), no. 3 1701610–1701610.

[90] S. Nagasawa, H. Numata, Y. Hashimoto, and S. Tahara, *High-frequency clock operation of Josephson 256-word/spl times/16-bit RAMs*, *IEEE Transactions on Applied Superconductivity* **9** (1999), no. 2 3708–3713.

[91] N. Yoshikawa, T. Tomida, M. Tokuda, Q. Liu, X. Meng, S. Whiteley, and T. Van Duzer, *Characterization of 4 k cmos devices and circuits for hybrid josephson-cmos systems*, *IEEE Transactions on Applied Superconductivity* **15** (2005), no. 2 267–271.

[92] K. Fujiwara, Q. Liu, T. Van Duzer, X. Meng, and N. Yoshikawa, *New delay-time measurements on a 64-kb josephson–cmos hybrid memory with a 600-ps access time*, *IEEE Transactions on Applied Superconductivity* **20** (2010), no. 1 14–20.

[93] T. Van Duzer, L. Zheng, S. R. Whiteley, H. Kim, J. Kim, X. Meng, and T. Ortlepp, *64-kb hybrid Josephson-CMOS 4 Kelvin RAM with 400 ps access time and 12 mw read power*, *IEEE Transactions on Applied Superconductivity* **23** (2013), no. 3 1700504–1700504.

[94] A. K. Feofanov, V. A. Oboznov, V. V. Bol'ginov, J. Lisenfeld, S. Poletto, V. V. Ryazanov, A. N. Rossolenko, M. Khabipov, D. Balashov, A. B. Zorin, and et al., *Implementation of superconductor/ferromagnet/ superconductor π-shifters in superconducting digital and quantum circuits*, *Nature Physics* **6** (Jun, 2010) 593–597.

[95] I. V. Vernik, V. V. Bol'ginov, S. V. Bakurskiy, A. A. Golubov, M. Y. Kupriyanov, V. V. Ryazanov, and O. A. Mukhanov, *Magnetic Josephson junctions with superconducting interlayer for cryogenic memory*, *IEEE Transactions on Applied Superconductivity* **23** (2013), no. 3 1701208–1701208.

[96] B. Baek, W. H. Rippard, S. P. Benz, S. E. Russek, and P. D. Dresselhaus, *Hybrid superconducting-magnetic memory device using competing order parameters*, *Nature Communications* **5** (May, 2014).

[97] E. C. Gingrich, B. M. Niedzielski, J. A. Glick, Y. Wang, D. L. Miller, R. Loloee, W. P. Pratt Jr, and N. O. Birge, *Controllable 0–π Josephson junctions containing a ferromagnetic spin valve*, *Nature Physics* **12** (Mar, 2016) 564–567.

[98] M.-H. Nguyen, G. J. Ribeill, M. V. Gustafsson, S. Shi, S. V. Aradhya, A. P. Wagner, L. M. Ranzani, L. Zhu, R. Baghdadi, B. Butters, E. Toomey, M. Colangelo, P. A. Truitt, A. Jafari-Salim, D. McAllister, D. Yohannes, S. R. Cheng, R. Lazarus, O. Mukhanov, K. K. Berggren, R. A. Buhrman, G. E. Rowlands, and T. A. Ohki, *Cryogenic memory architecture integrating spin hall effect based magnetic memory and superconductive cryotron devices*, *Scientific Reports* **10** no. 1.

[99] A. Murphy, D. V. Averin, and A. Bezryadin, *Nanoscale superconducting memory based on the kinetic inductance of asymmetric nanowire loops*, *New Journal of Physics* **19** (jun, 2017) 063015.

[100] Q.-Y. Zhao, E. A. Toomey, B. A. Butters, A. N. McCaughan, A. E. Dane, S.-W. Nam, and K. K. Berggren, *A compact superconducting nanowire memory element operated by nanowire cryotrons*, *Superconductor Science and Technology* **31** (feb, 2018) 035009.

[101] B. A. Butters, R. Baghdadi, M. Onen, E. A. Toomey, O. Medeiros, and K. K. Berggren, *A scalable superconducting nanowire memory cell and preliminary array test*, *Superconductor Science and Technology* **34** (jan, 2021) 035003.

[102] Y. Kameda, S. Yorozu, and Y. Hashimoto, *A new design methodology for single-flux-quantum (SFQ) logic circuits using passive-transmission-line (PTL) wiring*, *IEEE Transactions on Applied Superconductivity* **17** (2007), no. 2 508–511.

[103] P. Rux, *A glass delay line content-addressed memory system*, *IEEE Transactions on Computers* **C-18** (1969), no. 6 512–520.

[104] Y. Hashimoto, S. Yorozu, Y. Kameda, A. Fujimaki, H. Terai, and N. Yoshikawa, *Design and investigation of gate-to-gate passive interconnections for SFQ logic circuits*, *IEEE Transactions on Applied Superconductivity* **15** (2005), no. 3 3814–3820.

[105] S. K. Tolpygo, E. B. Golden, T. J. Weir, and V. Bolkhovsky, *Inductance of superconductor integrated circuit features with sizes down to 120 nm*, *Superconductor Science and Technology* **34** (jun, 2021) 085005.

[106] T. M. Hazard, A. Gyenis, A. Di Paolo, A. T. Asfaw, S. A. Lyon, A. Blais, and A. A. Houck, *Nanowire superinductance fluxonium qubit*, *Phys. Rev. Lett.* **122** (Jan, 2019) 010504.

[107] D. Niepce, J. Burnett, and J. Bylander, *High kinetic inductance* NbN *nanowire superinductors*, *Phys. Rev. Applied* **11** (Apr, 2019) 044014.

[108] W. Hattori, T. Yoshitake, and S. Tahara, *A reentrant delay-line memory using a* $YBa_2Cu_3O_{7-\delta}$ *coplanar delay-line*, *IEEE Transactions on Applied Superconductivity* **9** (1999), no. 2 3829–3832.

[109] A. Goda, *3-d nand technology achievements and future scaling perspectives*, *IEEE Transactions on Electron Devices* **67** (2020), no. 4 1373–1381.

[110] L. Lamport, *The temporal logic of actions*, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **16** (1994), no. 3 872–923.

[111] M. Benedetti and A. Cimatti, *Bounded model checking for past ltl*, in *Tools and Algorithms for the Construction and Analysis of Systems* (H. Garavel and J. Hatcliff, eds.), (Berlin, Heidelberg), pp. 18–33, Springer Berlin Heidelberg, 2003.

[112] F. Laroussinie, N. Markey, and P. Schnoebelen, *Temporal logic with forgettable past*, in *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pp. 383–392, 2002.