# High-performance cryogenic computing using superconductor single flux quantum logic

石田, 浩貴

出版情報：九州大学, 2020, 博士（工学）, 課程博士
バージョン：
権利関係：

# High-Performance Cryogenic Computing Using Superconductor Single Flux Quantum Logic

**Koki Ishida**

**A DISSERTATION**

**Kyushu University**

**March, 2021**

# High-Performance Cryogenic Computing Using Superconductor Single Flux Quantum Logic

Koki Ishida

## Abstract

Processors, the essential part of computer systems, have dramatically improved their performance with transistors' shrinking technology since the first processors are developed in the early 1970s. However, around the 2000s, the problem of power consumption became apparent, and it became hard to improve the clock frequency, which significantly contributed to the improvement of performance. Although processors' performance improvement has been maintained by introducing multi-core designs, it will be challenging to improve chip performance only with conventional CMOS computing technology continuously after the end of transistors' scaling.

To solve these problems, this dissertation focuses on a cryogenic computing technology using superconductor single flux quantum (SFQ) logic that has both high speed and low power consumption. SFQ logic uses low-voltage impulse-shaped signals for logic operations, which allows the ultra-fast ($10^{-12}$s) and low-energy switching ($10^{-19}$J). Due to these potentials, several researchers have so far contributed to SFQ-related research, and several physical implementations, including processors, have successfully demonstrated at the outstanding frequency, e.g., several tens of GHz. However, this is the performance at the logic gate level in the circuit, and when converted to the processor's performance based on software execution, it remains at the same level as conventional CMOS processors. The fundamental problem existing behind the SFQ processors is the lack of architectural optimizations to exploit the full potential of SFQ devices. Moreover, although several circuits' demonstrations have successfully shown the device-level potential, few studies focus on the architectural unit designs to show the SFQ computing potential. Therefore, the following questions must be clearly addressed to show the SFQ computing's potential: (1) what architecture is promising for this technology, (2) how is the feasibility and effectiveness of the architecture, (3) how to evaluate the effective performance and power efficiency of the target designs.

To solve the problem, this dissertation firstly explores the architectural design space of SFQ processors to determine the basic design guidelines for the realization of high-performance SFQ processors. Specifically, we assume a simple processor (i.e., in-order scalar processor) and analyze the effect of two architectural parameters (i.e., instruction pipeline stages and bit-width) on performance and power consumption. As a result, the bit-parallel processing and gate-level deep pipeline structure are suitable for achieving high performance in SFQ processors. Moreover, the result indicates that SFQ processors must conceal almost all pipeline stalls to achieve high performance in such an ultra-deep pipelining.

Second, this dissertation designs and implements a 4-bit SFQ processor chip as a prototype to clarify our proposed architectural design guidelines' effectiveness and feasibility. As a result, we confirm the correct operation at 32 GHz of clock frequency with 6.5 mW of power consumption. Moreover, this dissertation has extended into the 64-bit processor based on these results and evaluated its power efficiency with the cooling overhead for keeping SFQ circuits at 4 kelvin. The power efficiency, including the cooling cost, is estimated to at most 7.1 GOPS/W, and our processor outperforms the CMOS processor model 7.8 times.

Third, this dissertation proposes and evaluates the SFQ accelerator design for neural network applications to show the real potential of SFQ computing. Specifically, this dissertation designs the basic architecture of the neural network accelerator based on the basic design guidelines for SFQ processors and develops an evaluation environment based on power performance modeling. Besides, we analyze the performance bottleneck and optimize the architecture. As a result of the evaluation, our design outperforms about 23 and 1.2 times higher performance and power efficiency with the cooling cost compared to a state-of-the-art CMOS accelerator, respectively.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Challenges in the conventional CMOS computing

Processors are essential and fundamental components of various computer systems, e.g., computer servers, personal computers, and embedded systems. The processors' performance has continuously improved thanks to Moore's Law, the observation in which the number of transistors in a chip doubles every 18 months. Until the early 2000s, the processors have achieved their performance improvements by mainly increasing its frequency. However, in the early 2000s, the device scaling became ineffective in increasing the chip frequency because of the negligible power dissipation, i.e., power wall problem.

To get around the power wall problem, architects have introduced relatively slow multi- or many-core designs to improve chip performance continuously. Nevertheless, these circumventions can suffer from the parallelization overhead and the increasing on-chip power consumption. Moreover, while device and manufacturing technologies have continued progressing, it seems challenging to maintain the transistors shrinking because of physical or economic reasons, i.e., the end of Moore's Law is coming. After the end of Moore's Law, we are running out of an effective option to improve computer systems' performance while maintaining its power budget. Therefore, we believe that it is the right time to actively exploit emerging device technologies with significant potentials and make a serious effort to improve their feasibility by resolving their limitations.

## 1.2 Cryogenic computing using superconductor single flux quantum logic and its challenges

Cryogenic computing, which is to run a computer device at extremely low temperatures, is an attractive direction to make a breakthrough for achieving sustainable improvement of computer systems in the post-Moore's era. Among several candidates, superconductor single-flux-quantum (SFQ) logic [49, 58] and its energy-efficient families [43, 84, 86, 89, 78] are highly promising solutions thanks to their ultra-fast speed and low-power consumption at 4 kelvin. SFQ logic uses low-voltage impulse-shaped signals for logic operations, which allows the ultra-fast ($10^{-12}$s) and low-energy switching ($10^{-19}$J). Because of these potentials, several researchers have so far contributed to SFQ-related research in various aspects, especially in the device and circuit area. There are physical implementations, including processors, that have successfully demonstrated at the outstanding frequency, e.g., several tens of GHz [87, 77, 64].

Although several circuits have successfully demonstrated at several tens of GHz circuit frequency, it is hard to say that these demonstrations have sufficiently shown SFQ computing's potential. For example, a state-of-the-art SFQ processor [64] operates at 50 GHz, the performance (i.e., throughput) is relatively low, e.g., 333 million instructions per second (MIPS). The fundamental problem existing behind the SFQ processors is the lack of architectural optimizations to exploit the full potential of SFQ devices. Due to its unique pulse-driven nature, SFQ logic requires completely different architecture designs from conventional CMOS technology. Therefore, the following questions must be clearly addressed to show the SFQ computing's potential: (1) what architecture is promising for this technology, (2) how is the feasibility and effectiveness of the architecture, (3) how to evaluate the effective performance and power efficiency of the target designs.

## 1.3 Thesis statement

To resolve these challenges, this dissertation first proposes the architectural design guidelines for SFQ logic. Next, this dissertation shows the feasibility and effectiveness of the architectural design guidelines with a prototype chip design. Finally, this dissertation designs the neural processing unit (NPU) as a case-study architecture design and evaluates its performance and power efficiency with the cooling cost of 4 kelvin.

## 1.4 Contributions

This dissertation makes the following contributions:

- This dissertation explores the architectural design space of SFQ processors and quantitively shows that the bit-parallel processing and gate-level deep pipeline structure are suitable for achieving high performance. Moreover, the result indicates that to achieve high performance by such an ultra-deep pipelining, SFQ processors must conceal almost all pipeline stalls. Even with the small pipeline stalls are occurred, the performance improvement degrades 63 times to 5.7 times. To resolve the problem, this dissertation proposes fine-grained multithreading execution for hiding all pipeline stalls caused by data and control hazards, i.e., this scheme enables processor interlock-free streaming execution.

- To clarify our proposed architectural design guidelines' effectiveness and feasibility, this dissertation designs and implements a 4-bit SFQ processor as a prototype and confirm the correct operation by real chip measurement. As a result, the prototype chip has successfully operated at 32 GHz with 6.5 mW. Moreover, to clarify the cooling overhead for keeping SFQ circuits at 4 kelvin, this dissertation evaluates the power efficiency, including a cryocooler cost. Specifically, this dissertation has extended into the 64-bit processor based on these results and evaluated its power efficiency with two types of the real cryocoolers' cost. The best power efficiency (i.e., calculated by its peak performance) with the cooling cost is estimated to at most 7.1 GOPS/W, and our processor outperforms the CMOS processor model 7.8 times. The result indicates that our SFQ processor based on proposed architectural design guidelines has the potential to achieve high performance and power efficiency even with the cooling cost.

- To show the real potential of SFQ computing, this dissertation proposes and evaluates the SFQ-based neural processing unit (NPU) architecture as a case study. Specifically, first, this dissertation implements and validates an SFQ-based NPU modeling framework. It is the first work to model and validate a model and simulator for SFQ-based architectures. Next, by using the tool, we identify critical challenges in architecting an SFQ-based NPU. Finally, we present *SuperNPU*, our example SFQ-based NPU architecture, which effectively addresses the challenges at the architectural level. Our evaluation shows that the proposed design outperforms a conventional state-of-the-art NPU by 23 times with comparable power efficiency, including extremely expensive cooling costs.

## 1.5 Dissertation Organization

This dissertation is organized into 6 chapters. Chapter 2 introduces the challenges of CMOS processors, prior work that tackles the problem with cryogenic computing, and clarifies our research field. Chapter 3 proposes the architectural design guidelines for SFQ processors with architectural design space exploration and quantitive evaluations. Chapter 4 prototypes the SFQ processor chip based on the proposed guidelines and evaluates its feasibility and effectiveness. Chapter 5 shows the real potential of SFQ computing by designing and evaluating SFQ-based NPU with a validated simulation framework. Chapter 6 concludes this dissertation.

# Chapter 2

# Background

## 2.1 Trend and challenges of CMOS processors

Processors had improved their performance since 1971, when the first processor, Intel 4004, was developed [8]. Fig. 2.1 shows the 48-year CMOS processors' trend [1]. Thanks to Dennard scaling [22] and Moore's Law [65], architects can place more logics and memories on the same sized chip and increase the frequency without increasing its power consumption. However, Dennard scaling broke down in the early 2000s because of the negligible power dissipation (i.e., leakage power), and the device scaling has been ineffective in increasing the chip frequency. As you can see in the Fig. 2.1, processors have been limited in their capacity for clock speed improvement since the early 2000s. With the limit of clock frequency improvement, it is hard to improve the single-thread performance significantly.

After the end of Dennard scaling, architects have introduced multi- or many-core designs to improve chip performance. Specifically, they replace the fast single-core processor with slow multiple processor cores to improve the multi-thread performance while suppressing its power consumption. To execute a program in parallel, programmers must explicitly describe where to be executed in parallel in their source code. Ideally, the chip performance improves in proportion to the number of cores in a chip. However, programs that cannot be executed in parallel have no benefit from the multi-core designs, i.e., the single-thread performance is still important. Moreover, if the end of Moore's Law comes, we cannot expect

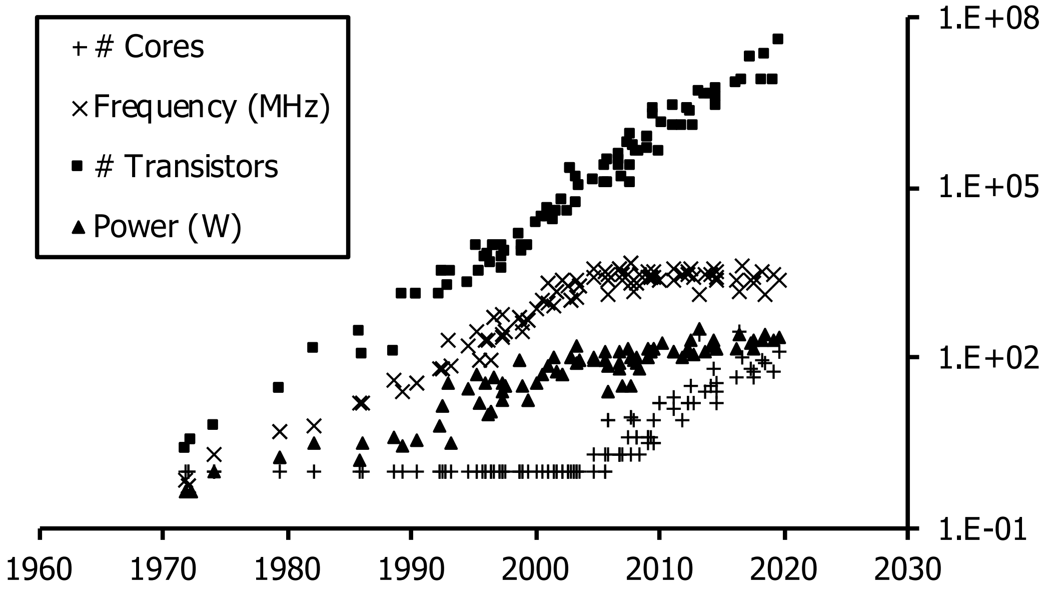


Figure 2.1: 48-year CMOS processors' trend

performance improvement by increasing the number of cores.

While device and manufacturing technologies have continued progressing, it does not seem easy to maintain the transistors shrinking because of physical or economic reasons. Fig. 2.2 shows the processor trends of Intel Xeon processors over generations [71]. The figure plots the number of threads running on a single core (i.e., SMT level), the number of cores integrated on a chip, and the package size of these processors. Increasing the SMT level is completely stopping because it requires much larger intra-core, memory-like architectural units (e.g., register files, load, and store queues, reorder buffer) to keep many architectural states, and it can decrease the number of cores in a chip. On the other hand, the number of cores is still increasing year after year. However, the package size is also increasing in proportion to the number of cores. There is a physical limit to relying on the increase of the package size, and even if the package can be larger, the total number of cores in the computer system will not change much. Thus, architects cannot put more cores on a chip without increasing the package size or reducing each core's size anymore. Therefore, we believe that it is the right time to actively exploit emerging device technologies with significant potentials and make a serious effort to improve their feasibility by resolving their limitations.

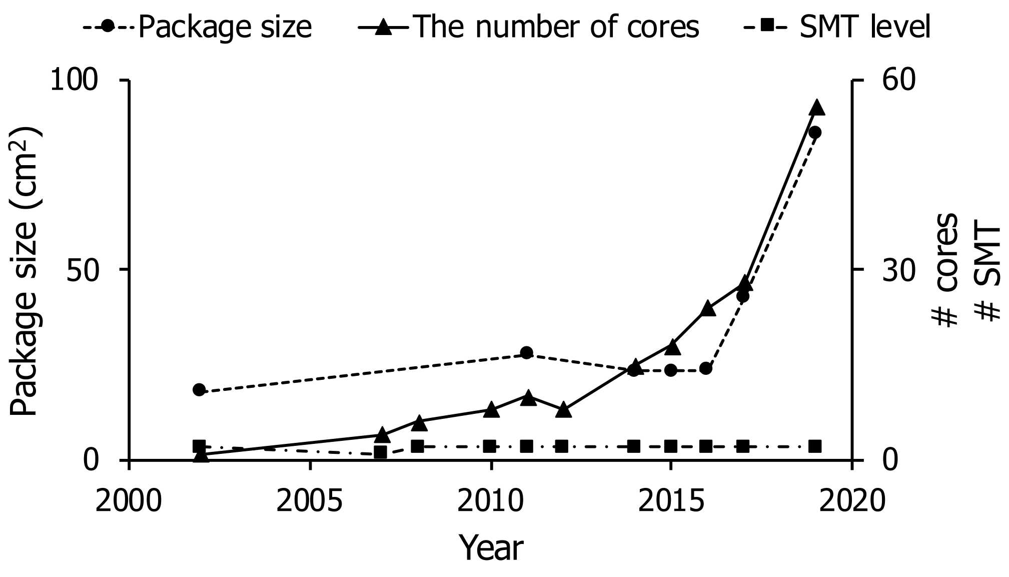


Figure 2.2: Trends of Intel Xeon processors over generations

## 2.2 Cryogenic computing

Cryogenic computing, which is to run a computer device at extremely low temperatures (e.g., 77, 4, or sub-millikelvin), is a promising approach for achieving sustainable improvement in the post-Moore's era. Such ultra-low temperatures allow introducing unique physical phenomena (e.g., superconductivity, quantum mechanics) for improving computer systems' performance. There are mainly three types of cryogenic computing: conventional CMOS computing at low temperature, superconducting computing, and quantum computing. These technologies are comparatively classified on two axes (i.e., performance and versatility) in Fig. 2.3.

### 2.2.1 Low-temperature CMOS computing

Cryogenic CMOS computing aims to improve both performance and power efficiency by reducing transistors' leakage current and wire resistance. If the leakage current decrease, we can apply a much higher clock frequency without increasing dynamic power consumption. Moreover, reducing wire resistance makes the wire latency lower, i.e., it becomes much easier to achieve high clock speed or low-latency signal transfer. Recent studies focusing on cryogenic memory, cache, and

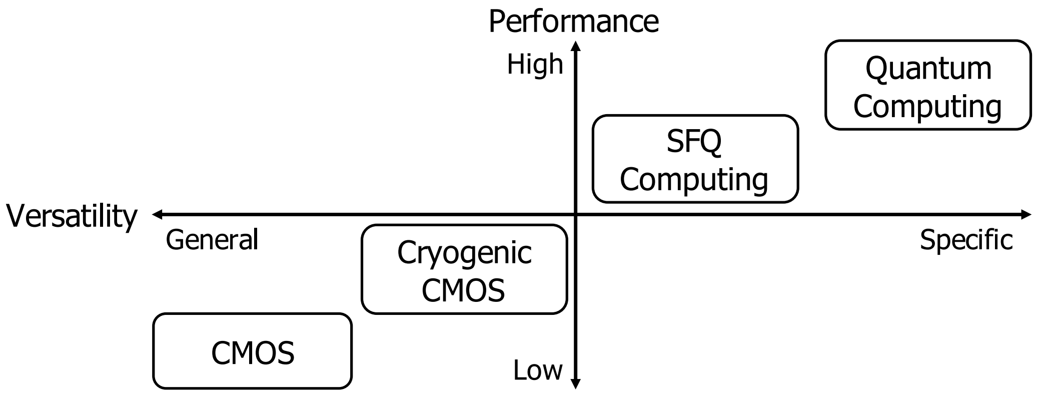


Figure 2.3: Classification of computing technologies

processor show the potential in both performance and power efficiency, including their cooling cost [47, 11, 53].

[47] reports the results of three promising case studies using cryogenic memories to significantly improve the server performance up to 2.5 times, decrease the server power to 6% on average, and reduce datacenter's power cost by 8.4%. [53] proposes cryogenic CMOS cache architecture that achieves 2 times faster cache access and 2 times larger capacity compared to conventional caches running at room temperature. [11] proposes two following types of cryogenic CMOS processor architectures. The high-performance design achieves 41% higher single-thread performance for the same power budget and 2times higher multi-thread performance for the same die area. The low-power design reduces power consumption by 38% without sacrificing the single-thread performance.

### 2.2.2 Superconducting computing

Superconducting computing uses superconductors' unique properties (e.g., zero-resistance wires and quantization of magnetic flux) for high performance and power efficiency computation. Superconducting circuits use Josephson junctions as basic elements and require cooling to 4 kelvin. There are several superconducting logics such as single-flux-quantum (SFQ) logic family [49, 58], reciprocal quantum logic (RQL) [33], and adiabatic quantum-flux-parametron (AQFP) logic [51].

SFQ logic family is the most practical superconducting VLSI technology and has ultrafast speed and low-power consumption natures. By focusing on these high potentials, many serious SFQ-related research efforts have been made in various aspects, and a lot of SFQ circuits and their successful operations have been demonstrated [23, 87, 77, 64, 56, 55, 24, 9, 80].

RQL and AQFP logics are relatively slow but more energy-efficient technology compared to SFQ logic. Both use alternating current for power supply instead of direct current, which is employed in SFQ logic. Therefore, they achieve more energy efficiency with sacrificing speed compared to SFQ logic. There are several circuit demonstrations and architecture consideration studies [12].

### 2.2.3 Quantum computing

Quantum computing is entirely different from conventional digital (or classical) computing and uses quantum bits, or qubits, to encode information as logical '0', '1', or both at the same time. This superposition of states enables quantum computers to manipulate enormous combinations of states at once. Therefore, quantum computing has the potential to solve problems that conventional computers could not solve in a realistic amount of time.

Real implementations that use a process called quantum annealing are available in the market as quantum computers [30, 38]. Although they can effectively be applied to specific purposes such as quantum annealing, there is a large gap regarding functionality between classical digital computing and the application-specific quantum acceleration. Therefore, there are several studies about more general-purpose quantum computing using quantum logic gates, and several companies try to build the first practically useful quantum computer [20, 27, 66].

## 2.3 Superconductor single flux quantum logic

Superconductor SFQ logic [49, 58] and its energy-efficient families [43, 84, 86, 89, 78] are representative ultra-fast and low-power VLSI technologies using superconducting devices, namely Josephson junctions (JJs). This dissertation focuses on the SFQ logic technology because it is one of the most practical technology with emerging device potential for the next-generation computer. In this section, first,

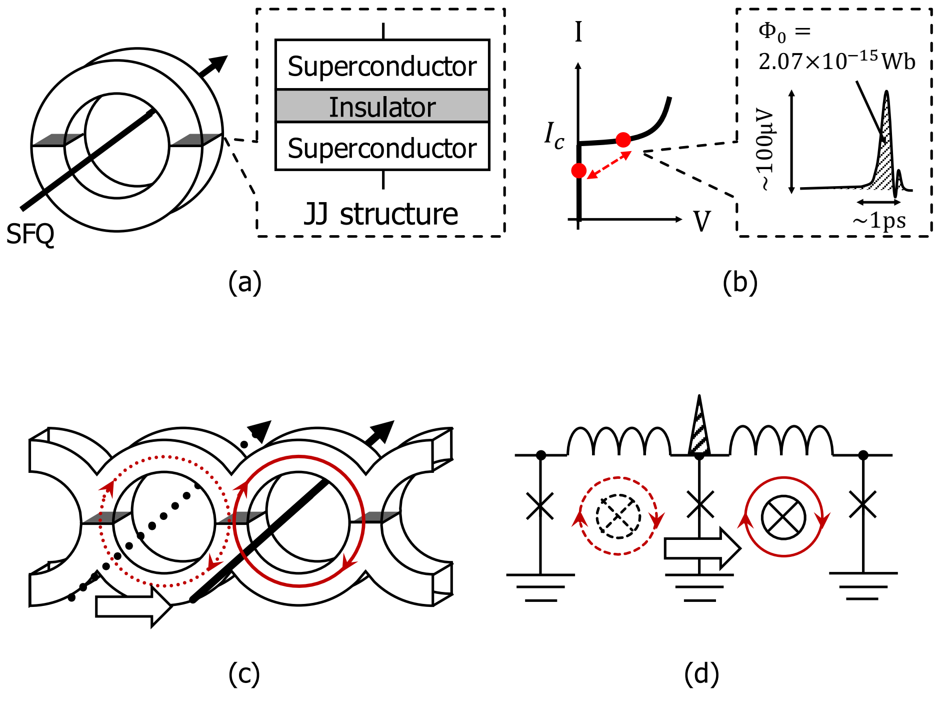


Figure 2.4: (a) Superconductor ring with SFQ (b) Electrical characteristics of JJ (c) Serially connected SFQ rings and (d) its equivalent circuit diagram

the working principle of SFQ circuits is introduced. Next, this dissertation explains the frequency determination and power calculation of SFQ circuits based on pulse-driven logic.

### 2.3.1 Basic elements of SFQ circuit

Fig. 2.4(a) shows a basic circuit element of SFQ technology, a superconductor ring. SFQ circuits utilize the existence of a single magnetic flux quantum (SFQ) in the superconductor ring as an information carrier, similar to the voltage level in conventional CMOS circuits. Specifically, the presence or absence of an SFQ in the ring represents a logical '1' or '0'. The superconductor ring can store and transfer the SFQ by using a superconducting device called Josephson junction (JJ). In this dissertation, we use the superconductor-insulator-superconductor (SIS) structured JJs, as shown in Fig. 2.4(a), that use niobium (Nb) as the superconductor and aluminum oxide (AlOx) as the insulator layer. The JJ included in the ring acts as a switching device like a transistor and JJs' electrical characteristic is shown

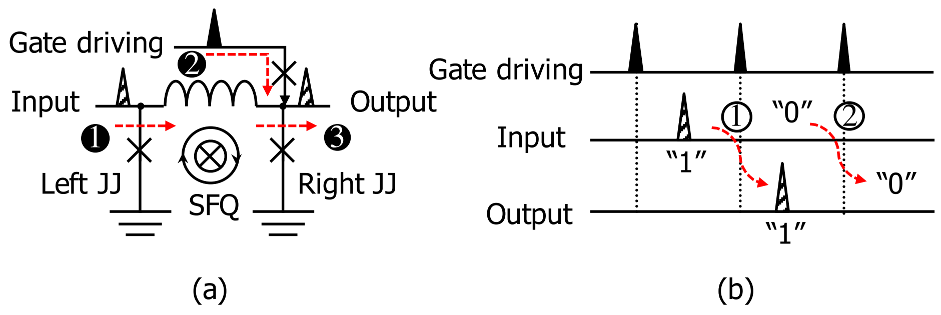


Figure 2.5: (a) Circuit diagram of an SFQ-based DFF with (b) its operating example

in Fig. 2.4(b). If the current flowing JJ exceeds its critical current $I_c$, the JJ will switch, and an impulse-shaped voltage pulse, called an SFQ pulse, will be generated. SFQ pulses have a quantized area $\Phi_0$ due to magnetic flux quantization, a fundamental property of superconductors, as shown in Eq. (2.1).

$$\int \mathrm{V}(t)dt = \Phi_0 \simeq 2.07 \times 10^{-15}\mathrm{Wb} = 2.07\mathrm{mV} \cdot \mathrm{pH} \tag{2.1}$$

As shown in Fig. 2.4(b), the typical width and height of the SFQ pulse are a few picoseconds and a few hundreds of microvolts, respectively. The SFQ pulse allows the ultra-fast ($10^{-12}$s) and low-energy switching ($10^{-19}$J).

Fig. 2.4(c) shows the serially connected SFQ rings and Fig. 2.4(d) shows its equivalent circuit diagram. Inductance and cross marks represent the superconductor part of the ring and JJs, respectively. These serially connected SFQ rings are used to one of the SFQ wirings named Josephson transmission line (JTL). The signal is propagated by switching the ring in order from the left in JTL. Besides, SFQ signals can be split or merged using a few JJs, and easily stored in large inductance superconducting rings, i.e., the ring can play the role of delay flip flops (DFF). All SFQ gates or wire cells are composed of a combination of the SFQ rings, and the working principle of the SFQ gates are explained in Section 2.3.2.

## 2.3.2 Working principle of SFQ circuit

Unlike voltage level logic, synchronization of an input signal with the reference signal (from now on *gate driving pulse*) is essential at a storage ring in each SFQ logic gate to distinguish between that the input signal has not arrived yet and

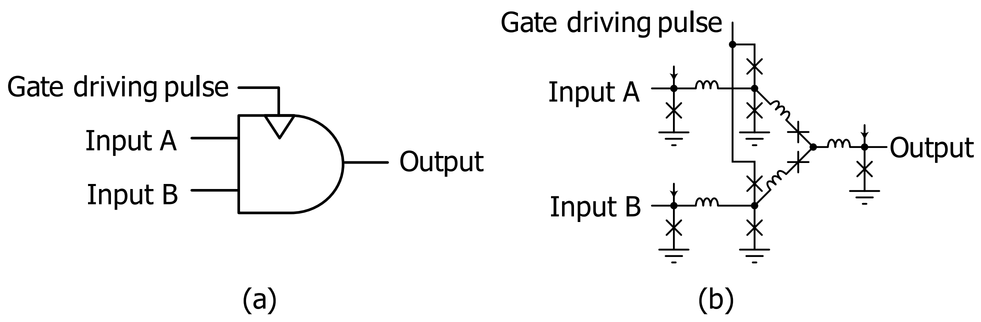


Figure 2.6: (a) SFQ-based AND gate with (b) its circuit diagram

that a logic value ‘0’ has arrived. In other words, each storage ring intrinsically has latch or memory functionality, and all the SFQ logic gates are clocked gates except for wire cells such as pulse splitters and mergers.

First, an SFQ-based DFF is taken as our example due to its simplest structure consisting of only a single superconductor ring and a gate driving pulse line. When the input pulse comes to the ring, it makes the current flowing through the left JJ of the ring higher than its critical current, $I_c$. With the electrical characteristic shown in Fig. 2.4(b), the left JJ generates a voltage pulse and it is stored to the ring as an SFQ (Fig. 2.5(a) ❶). Next, by taking a gate driving pulse (Fig. 2.5(a) ❷), the right JJ is activated and the stored SFQ is transferred to the output as a voltage pulse (Fig. 2.5(a) ❸). In this manner, SFQ gates can define the logical value ‘1’ as the existence of stored SFQ between the gate driving pulses (Fig. 2.5(b) ①). On the other hand, if no input pulse comes during a gate driving pulse period, no voltage pulse is generated on the output, and it indicates the logical value ‘0’ (Fig. 2.5(b) ②).

Next, we introduce how to build SFQ-based AND gate with its operating examples. Fig. 2.6 shows the SFQ-based AND gate and its circuit diagram. As mentioned in Section 2.3.1, SFQ logic gates are composed of a combination of the SFQ rings, and AND gate is mainly composed of three SFQ rings. Fig. 2.7 shows the operating examples of SFQ AND gate. When the input pulses come to the AND gate (Fig. 2.7(a)), these signals are stored as SFQ in the AND gate’s SFQ rings (Fig. 2.7(b)). Next, by taking a gate driving pulse, the JJs of the SFQ rings storing inputs are activated, and the stored SFQ are transferred to the output

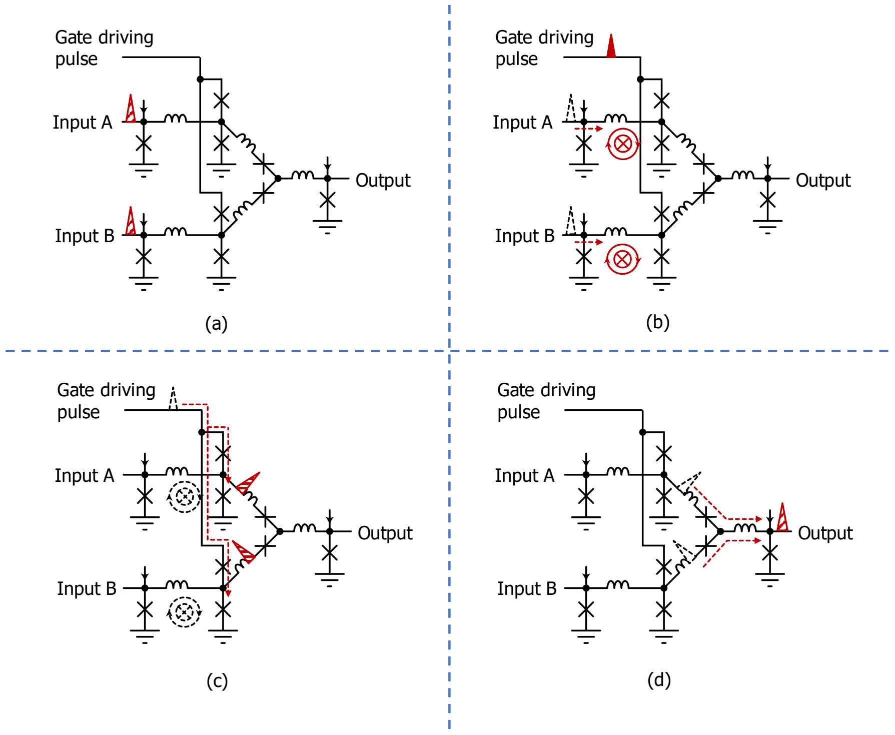


Figure 2.7: Operating examples of SFQ AND gate

as voltage pulses (Fig. 2.7(c)). The two data pulses switch the JJ at the output port, and the logical AND operation is performed. The JJ at the output port only switches when both two SFQ rings output data pulses.

Besides, we explain how to operate SFQ logic gates with serially connected DFFs, as shown in Fig. 2.8. In this circuit, gate A and gate C have an SFQ inside their ring initially; on the other hand, gate B has no data (Fig. 2.8 (a)). First, a gate driving pulse arrives at gate A and an SFQ stored in gate A outputs as a data pulse (Fig. 2.8 (b)). Second, the gate driving pulse arrives at gate B (Fig. 2.8 (c)). However, there is no output from gate B because gate B does not have an SFQ in its ring. To guarantee the correct operation, gate B must get the gate driving pulse before the data pulse input. In the timing design of SFQ circuits, designers must adjust the arrival timing of the data pulse and gate driving pulse. For example, inserting the delay elements in the data path can satisfy such timing constraints. Finally, gate C gets the gate driving pulse and outputs the data pulse (Fig. 2.8 (d)). In this way, the SFQ circuits consisting of memory functional logic gates operate with gate driving signals.

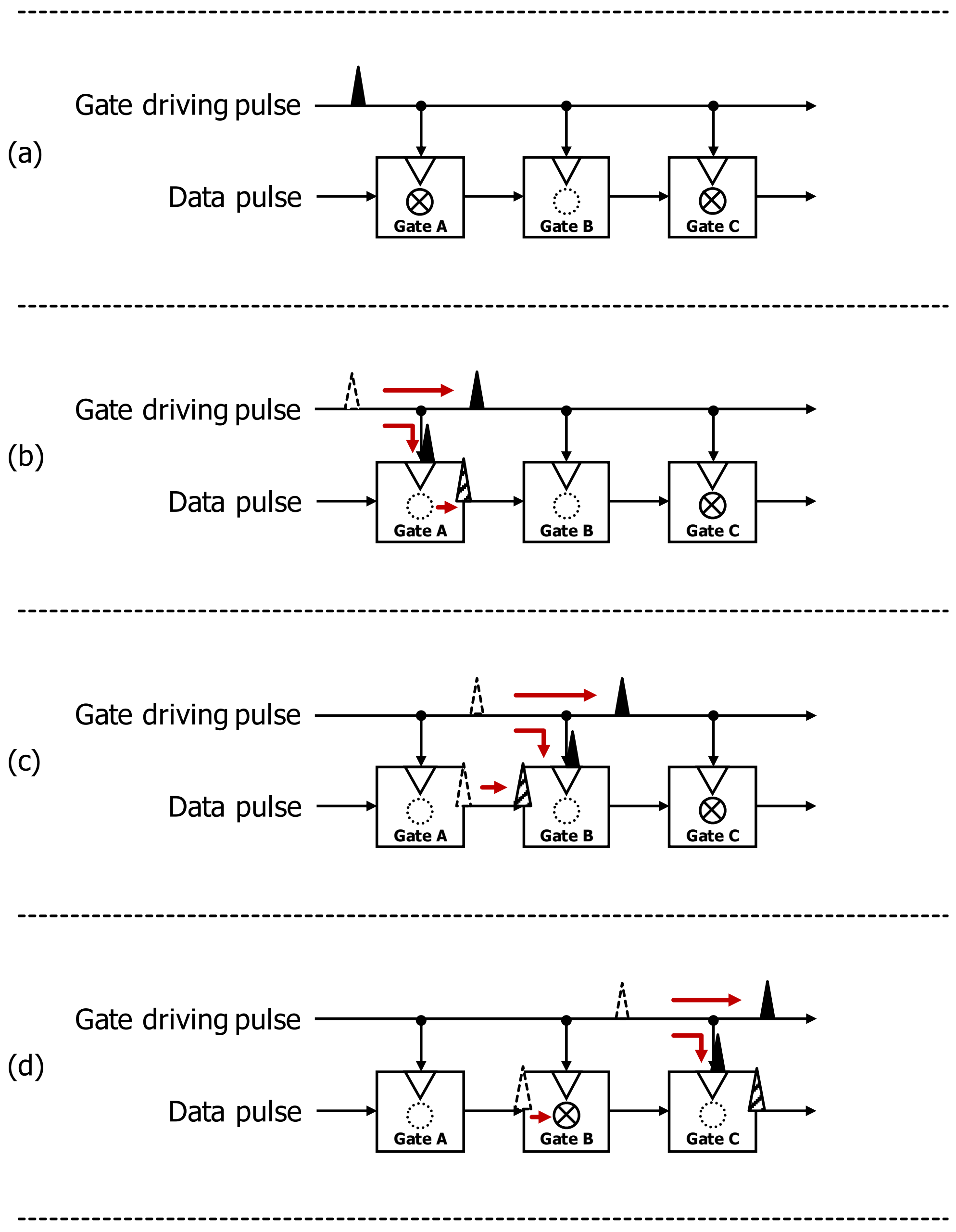


Figure 2.8: Operating example of the serially connected DFFs

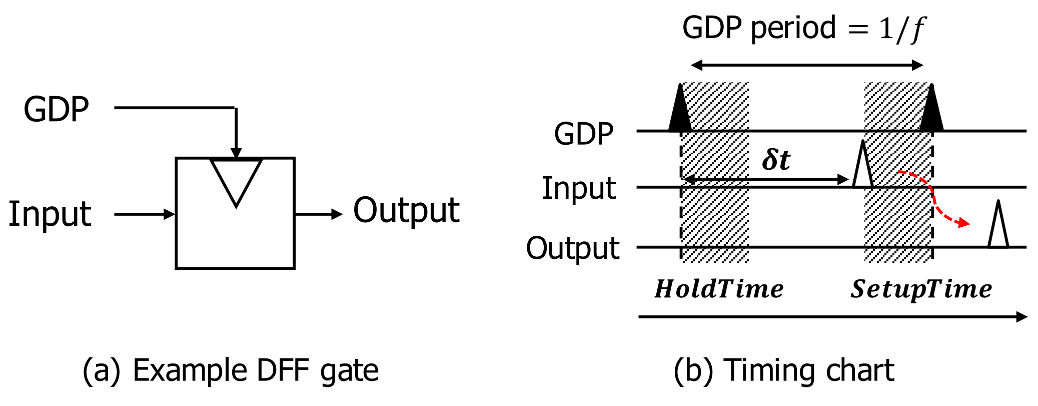


Figure 2.9: Example timing chart of an SFQ DFF gate

## 2.3.3 Frequency determination

Unlike conventional CMOS technology, SFQ circuits' frequency is determined by the timing difference between the data and gate driving pulse (GDP) arrival and timing constraints of origin and destination gates. In the CMOS technology, the clock frequency is bounded by the longest datapath delay because it only can put single digital information (i.e., voltage level) in a wire. On the other hand, SFQ logic can put several data into a single wire because its data is encoded as a voltage pulse. That pulse encoding enables SFQ circuits to flow many data pulses through a single wire simultaneously. Therefore, the critical factor in determining SFQ circuits' frequency is the difference between the arrival timing of data and gate driving pulse, not the wire length.

As mentioned in Section 2.3.1, SFQ logic gates have inherently memory or latching functionality. In other words, each SFQ logic gate has its own two types of timing constraints, *HoldTime* and *SetupTime*. Data from the origin gate should arrive at the destination gate after the HoldTime of the destination gate, and the next GDP should arrive after SetupTime elapsed from the data arrival. If the input pulse violates the HoldTime or SetupTime, the pulse will not be recognized as an input of logical '1'. Fig. 2.9 shows the example timing chart of an SFQ DFF gate. The circuit frequency is calculated by following Eq. (5.1).

$$f = 1/\text{GDP cycle time} = 1/(\mathit{SetupTime} + \text{Max}(\mathit{HoldTime}, \delta t)) \tag{2.2}$$

$\delta t$ represents the timing difference between the data and GDP arrival and it depends on the circuit structure.

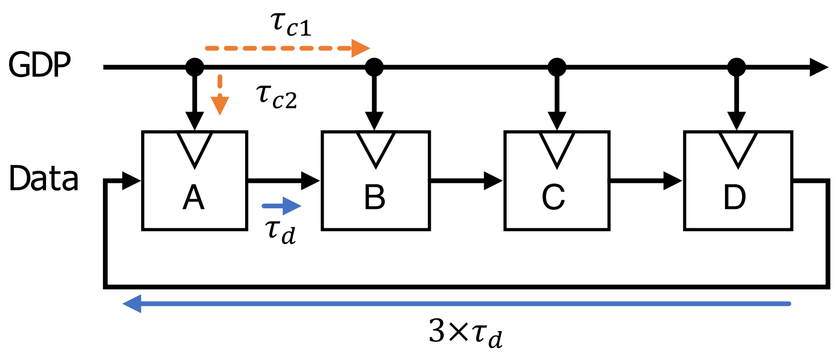
One of the most influential factors on the $\delta t$ is how to provide GDP (from now on called the clocking scheme). SFQ circuits employ flow clocking schemes where the GDP signals also propagate inside the circuits, i.e., not all but part of SFQ logic gates are synchronized [49]. Thus, the clocking scheme is an essential factor to determine the GDP frequency of SFQ circuits. There are mainly two clocking schemes in SFQ logic designs; the concurrent-flow (Fig. 2.10(a)) and the counter-flow clocking (Fig. 2.10(b)). In the concurrent-flow clocking, the GDP and data pulses propagate in the same direction. On the other hand, the GDP and data pulses propagate in the opposite direction in the counter-flow clocking.

To clarify the difference between the two clocking schemes, we calculate $\delta t$ of each clocking scheme. As shown in Fig. 2.9, $\delta t$ is calculated by Eq. (2.3)

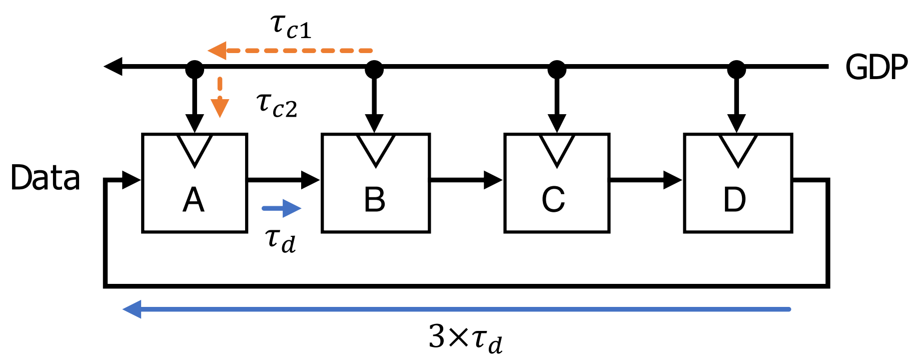
$$\delta t = T_{data} - T_{GDP} \tag{2.3}$$

, where $T_{data}$ and $T_{GDP}$ are the arrival timing of data pulse and GDP, respectively. Both $\delta t$ of concurrent-flow clocking and counter-flow clocking are summarized in Fig. 2.10(c). In the concurrent-flow clocking (Fig. 2.10(a)), the $T_{data}$ of gate B is calculated to $\tau_{c2} + \tau_d$. On the other hand, $T_{GDP}$ of gate B is represented in $\tau_{c1} + \tau_{c2}$. Thus, the $\delta t$ of gate B is $\tau_d - \tau_{c2}$, and $\delta t$ can be minimized by maching the data pulse and GDP propagation delay. In other words, the concurrent-flow clocking can achieve higher circuit frequency because it can hide the data propagation delay by flowing the GDP and the data pulse. However, such clocking cannot be utilized when the circuit includes the feedback loop, where the GDP and data pulses propagate in the opposite direction. Because gate A should wait for the output of gate D in Fig. 2.10(a), $T_{data}$ of gate A is calculated to $3\tau_{c1} + \tau_{c2} + 3\tau_d$. On the other hand, $T_{GDP}$ is represented in $\tau_{c2}$. As a result, $\delta t$ of gate A is $3\tau_{c1} + 3\tau_d$ and it increases in propotion to the depth of feedback loop. The circuit's frequency can be significantly reduced because the next GDP should wait for a very long data transfer through the feedback path.

On the other hand, this problem can be resolved with the counter-flow clocking, which can entirely hide the data feedback delay. Specifically, $\delta t$ of gate A is represented in $3\tau_d - 3\tau_c$ and the long data transfer through the feedback path can be hidden. However, the counter-flow circuit's frequency is much lower than that



(a) Concurrent flow



(b) Counter flow

| | $\Delta\tau$ w/o FB | $\Delta\tau$ w/ FB |
|---|---|---|
| Concurrent flow | $\tau_d - \tau_c$ | $3\tau_d + 3\tau_c$ |
| Counter flow | $\tau_d + \tau_c$ | $3\tau_d - 3\tau_c$ |

(c) Comparison of $\Delta\tau$

Figure 2.10: Illustration of flow clocking schemes (a) Concurrent-flow clocking (b) Counter-flow clocking (c) Comparison of $\delta t$

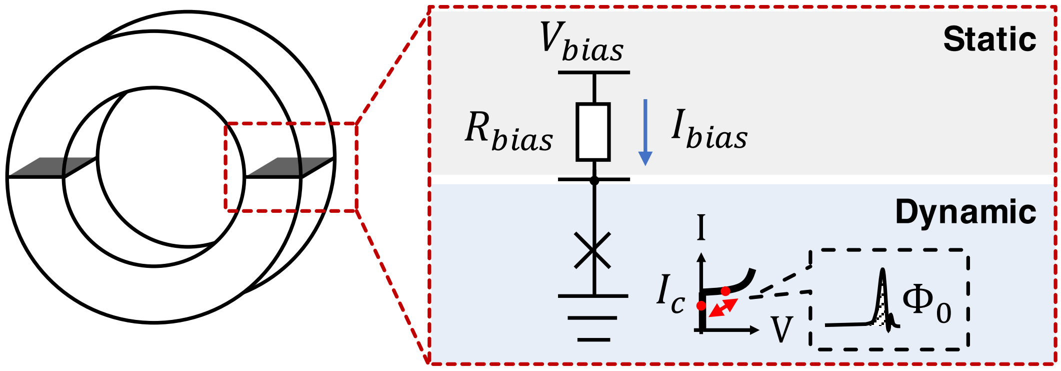


Figure 2.11: Circuit diagram of a biased JJ

of the concurrent-flow circuit without feedback. This is because the GDP and data pulses propagate in the opposite direction in the feedforward path (i.e., $\delta t$ is $\tau_d - \tau_{c1}$). Therefore, the existence of a feedback loop is the key factor to determine the clocking scheme.

## 2.3.4 Power consumption

The power consumption of the SFQ circuit consists of the static power consumed by the bias current supply and the dynamic power consumed when the JJs are activated. Fig. 2.11 shows a circuit diagram of a JJ biased by direct current. The total power consumption of the SFQ circuit $P_{total}$ is given by Eq. (2.4).

$$P_{total} = P_{static} + P_{dynamic} \tag{2.4}$$

where $P_{static}$ and $P_{dynamic}$ is the static and dynamic power consumption, respectively. As shown in Fig. 2.11, the bias current is continuously supplied to the JJ in the SFQ circuit, and it consumes the power regardless of JJ's switching. The static power $P_{static}$ is calculated by Eq. (2.5).

$$P_{static} = V_{bias} I_{bias} \times N_{JJ} \tag{2.5}$$

where $V_{bias}$ and $I_{bias}$ is the bias voltage and bias current, respectively. $N_{JJ}$ is the number of JJs included in the target circuit.

The dynamic power consumption in SFQ circuits is caused only during JJ's switching event; otherwise, the JJs are in the zero-voltage state (Fig. 2.4(b)), and no energy is consumed. When the JJ switches, the current flowing JJ exceeds its critical current $I_c$ and generates SFQ pulse, that area corresponds to $\Phi_0$. Thus, the dynamic energy for one JJ's switching event is calculated by Eq. (2.6).

$$E_{dynamic} = I_c \Phi_0 \tag{2.6}$$

Besides, the dynamic power is given by Eq. (2.7).

$$P_{dynamic} = E_{dynamic} \times N_{activeJJ/s} \tag{2.7}$$

where $N_{activeJJ/s}$ is the number of activated JJs per second in the target circuit.

## 2.4 Current status and research trend of SFQ technology

Due to the SFQ logic's high potentials, many serious SFQ-related research efforts have been made in various aspects to promote the technology. This dissertation introduces the current research in fabrication process technology, energy-efficient SFQ logic technology, circuit demonstrations, and SFQ memory technology in the following subsections.

### 2.4.1 Fabrication process technology

In conventional CMOS technology, "technology node" or "feature size" represent the minimum line width or the smallest machining dimension of CMOS transistors. On the other hand, SFQ logic technology's feature size represents the side length of Josephson junctions. The National Institute of Advanced Industrial Science and Technology (AIST) in Japan provides the AIST 1.0 $\mu$m Niobium (Nb) process technology for chip fabrication [57]. There are several fabricated chips for SFQ circuit demonstration by using this fabrication process technology. For instance, the state of the art bit-serial 8-bit processor, CORE e2, which successfully operates at 50 GHz, consists of 10,604 JJs [64]. In the 8-bit multiplier design, 20,251 JJs are integrated on a 6.03 mm $\times$ 5,22 mm chip area [55]. With this technology, the chip can integrate a few tens of thousands of JJs. It is possible to design prototypes of element circuits such as adders, multipliers, and processors to demonstrate the concept of architecture.

However, it is hard to design fully functional or practical units due to the low JJs' integration density. The AIST fabrication technology is relatively large compared to CMOS technology because the AIST uses an i-line stepper with a wavelength of 365 nm introduced to the market in the mid-1990s. The state-of-the-art steppers using KrF or ArF excimer lasers would allow the fabrication of ultrafine Josephson junctions and patterns. To the best of our knowledge, there is no study that mentions the physical limit of JJ scaling. On the other hand, there is the scaling rule that the frequency increases in proportion to the reduction rate of JJ until 200˜300 nm [40], and T flip-flop (TFF) has successfully demonstrated at up to 770 GHz with the technology [15]. Moreover, there are several schemes to reduce the SFQ cell size without the JJ scaling, such as the introduction of a shunt-resistor-free junctions [41], vertically-stacked junctions [13], multi-layer process technology with high-inductance layers [82], and new materials, such as niobium nitride. It is expected that process manufacturing technology will advance, and the device performance will improve in the future.

### 2.4.2 Energy-efficient SFQ logic technology

To ensure a more advantage in energy efficiency towards CMOS technology even with the cooling cost for 4 kelvin, researchers have so far contributed to developing energy-efficient circuit technologies. SFQ circuits were biased using direct current sources supplied by the external voltage sources with on-chip bias resistors. The total power of SFQ circuits is dominated by Joule heating in bias resistors (i.e., static power) rather than the JJ switching (i.e., dynamic power). Therefore, several studies are focusing on reducing SFQ circuit static power.

Inductance-load biasing [86], called LR biasing, replaces a bias resistor into a large inductor with a small resistor to reduce the static power. On the other hand, energy-efficient rapid SFQ (ERSFQ) logic technology [43, 84, 86, 89, 78] has successfully eliminated the static power. The only difference from standard SFQ gates is replacing bias resistors with the limiting JJs and series inductances. No major redesign of the SFQ logic gates is required in both technologies.

### 2.4.3 Circuit demonstrations

FLUX-1, the first single-chip SFQ processor, has been designed and fabricated in 2001s [23]. FLUX-1 chip represents an 8-bit bit-parallel processor prototype with a target GDP frequency of 17-20 GHz. This chip contains 65,759 JJs on a 10.6 mm $\times$ 13.2 mm die with flip-chip packaging. Unfortunately, the operation verification of the chip has failed.

Core 1 $\alpha$, the first successfully operated SFQ processor, has been designed and fabricated in 2004s [79]. This processor has employed 8-bit bit-serial processing for successful demonstration and operated at 15 GHz with a power consumption of 1.6 mW. This chip consists of 4,999 JJs. Other bit-serial processors have been designed and successfully demonstrated its correct operation [77, 87, 64]. State of the art bit-serial processor, CORE e2, operates at 50 GHz [64]. This processor includes 10,604 JJs and consumes 2.52 mW.

As described above, the bit-serial operation leads to a lot of successful circuit demonstrations. This is because bit-serial processing reduces hardware and long-distance wiring, and the timing adjustment of SFQ pulses becomes easy. However, such bit- by-bit fine-grained operations make the execution time much longer, resulting in poor performance. Therefore, it is a straightforward next step to consider the architecture exploiting the full potential of SFQ devices.

### 2.4.4 Memory technology

**On-chip memory technology**: For the SFQ logic's on-chip memory, the shift-register-based memory is much more practical than the random access memory (RAM). Even though we can implement RAM with SFQ technology, it severely suffers from low driving capability and scalability. Such limitations mainly result from the difficulty of driving the word lines and bit lines with the small pulses [49, 88]. On the other hand, a shift-register-based memory does not have those problems because it just consists of the serially connected DFFs and the feedback loop. However, it is difficult for the shift-register-based memory to support the random memory access due to the complex control logic and the variable access latency [37]. Therefore, the SFQ technology favors applications with sequential memory access when its on-chip memory implementation is considered.

**Off-chip memory technology**: It has been a long-standing challenge to implement a large-scale and high-speed off-chip memory operating at the 4K environment. There has been a few research about JJ-based memories [73, 45, 76], and one of them is Vortex Transition Memory (VTM) [73]. The VTM is the largest Josephson memory whose 4-kbit prototype has been demonstrated. Despite the demonstration, it has been difficult to practically use the VTM mainly due to the scaling and speed problems with the AC-biasing and the large superconductor-ring-based memory cells. Even though several off-chip memory technologies (e.g., hybrid Josephson-CMOS memory [45, 76], Josephson magnetic memory [21]) are currently being developed, these technologies also have not been put to practical use yet. For these reasons, it is currently practical to use CMOS memory technology, which is slower than the 4-kelvin JJ-based memory but large and reliable. Therefore, SFQ technology favors computation-oriented applications with a minimal number of off-chip memory access.

# Chapter 3

# Exploring design space of a SFQ processor

## 3.1 Introduction

As mentioned in Chapter 2, researchers have so far greatly been contributed to developing SFQ devices and logic design technologies; their physical designs and the successful operations of SFQ microprocessors have been demonstrated [87, 77, 64]. A state-of-the-art SFQ processor [64] have successfully operated at 50 GHz of GDP frequency. However, its effective performance, i.e., throughput, is significantly low compared to its circuit frequency, e.g., 333 million instructions per second (MIPS).

The fundamental problem existing behind the SFQ processors is the lack of architectural optimizations to exploit the full potential of SFQ devices. Specifically, these processors employ bit-serial processing to reduce hardware and long-distance wiring, and the timing adjustment of SFQ pulses becomes easy. However, such bit- by-bit fine-grained operations make the execution time much longer, resulting in poor performance. Moreover, the pipeline structure of these SFQ processors is from a course-grained pipeline such as the traditional 5-stage pipeline that consists of fetch, decode, execution, memory access, and writeback stages, despite the difference of device characteristics. It is essential to revisit SFQ processor architecture that fully exploits its device potential towards realizing extremely high-performance SFQ processors.

To solve the problem, we clarify the conventional SFQ processors' problems and quantitively shows the design guidelines for high-performance SFQ processors by conducting architectural design space exploration. As a result, we show the bit-parallel processing with gate-level pipelining is suitable for high-performance SFQ processors. Moreover, we propose fine-grained multithreading execution as the pipeline stall concealment technologies not to degrade its performance.

## 3.2 Architectural design space of SFQ processors

### 3.2.1 Architecture paramaters

In this section, this dissertation introduces two key architecture parameters for determining SFQ processors' performance.

- **Pipeline depth**: Pipeline depth represents the number of pipeline stages. Whichever CMOS and SFQ circuits, it is essential in determining a processor's performance. First, a general five-stage pipeline structure is set as a standard. Those with a deeper pipeline are called super pipeline structures, improving the clock frequency by reducing one pipeline stage's delay. Ultimately, the most fine-grained pipeline structure, that a deeper pipeline to the gate level is also conceivable. Although the clock frequency can be improved by increasing the pipeline depth, there are drawbacks such as 1) area overhead of the pipeline registers and 2) increased power consumption due to the increased clock frequency. However, the SFQ logic gate has no such drawbacks. Since all SFQ logic gates have a latch function, it is unnecessary to add pipeline registers to deepen the pipeline. Moreover, the SFQ gates can operate with less than 1 / 1,000 power consumption than the case of CMOS gates [54]. By utilizing these fundamental features, it is possible to solve the above two problems. Therefore, in SFQ design, it is expected that the performance will be greatly improved by increasing the pipeline depth.

- **Datapath bit width**: The datapath bit width is also one of the architecture parameters and represents the bit width handled by each unit, such as an

ALU or register file in one process. The data word length is the data that can be handled by the processor. *Slice* corresponds to the data whose data word length is divided. For example, each 8-bit data obtained by dividing 64 bits into eight is called a slice. In this case, the number of slices is 8. Based on the above, the design space is defined in the same way as the pipeline depth. There are mainly three types of processing schemes, bit-serial, bit-slice, and bit-parallel processing. In the bit-serial and slice processing schemes, divided data are pipelines in the units (e.g., ALU or register file) at a bit or slice level. On the other hand, data can be overlapped at the word level in bit-parallel processing.

### 3.2.2 Performance model

In this section, to consider processor architecture suitable for SFQ circuits, we use a performance model to clarify each architecture parameter's effect on the SFQ processor performance. In this architectural design space exploration, the time per instruction (TPI) is used [31]. TPI is calculated by Eq. (3.1).

$$TPI = \frac{T}{N_I} = (\frac{t_o}{\alpha} + \gamma\frac{N_H}{N_I}t_p) + \frac{t_p}{\alpha p} + \gamma\frac{N_H}{N_I}t_o p \tag{3.1}$$

Each paramater is shown as follows.

- $T$: Total execution time.
- $N_I$: The number of total instructions.
- $N_H$: The number of total pipeline hazards.
- $t_o$: The delay for latching data at pipeline registers. It depends on the HoldTime and SetupTime.
- $t_p$: The total gate delay of the longest path in the pipeline.
- $p$: The number of pipeline stages.
- $\alpha$: The number of simultaneous issued instructions per cycle. If the $\alpha$ is 2, 2 instructions are issued at one clock cycle.

Table 3.1: Delay parameters

| | $t_o$ (ps) | $t_p$ (ps) |
|---|---|---|
| 1.0$\mu$m SFQ-BP | | 2517.76 |
| 1.0$\mu$m SFQ-BSE | 13.32 | 13232.8 |
| 1.0$\mu$m SFQ-BSL | | 4565.4 |
| 0.3$\mu$m SFQ-BP | | 755.328 |
| 0.3$\mu$m SFQ-BSE | 3.995 | 3969.84 |
| 0.3$\mu$m SFQ-BSL | | 1369.62 |
| CMOS-BP | 86.76 | 4048.58 |

- $\gamma$: The ratio of the average pipeline stall time per the latency for instruction execution, i.e., $t_o p + t_p$. The maximum value is 1, which corresponds to the situation where subsequent instructions cannot be started until the instruction existing in the first pipeline stage is committed. On the other hand, the minimum value is 0, which indicates the situation where none pipeline stalls occur. For simplicity, this model uses the average value as $\gamma$. Note that the pipeline stalls caused by various hazards vary depending on the microarchitecture and the hazard occurrence situation. In this model, it is expressed as an average value.

The first item of Eq. (3.1) in parentheses represents the execution time increase caused by stalls and $t_o$, which is independent of the number of pipeline stages. The second item shows the effect of pipeline depth and $\alpha$ on the latency for processing one instruction. The third item indicates the pipeline stall overheads.

### 3.2.3 Delay parameters setup

In this experiment, SFQ processors are assumed as scalar processors (i.e., $\alpha = 1$), and we evaluate SFQ-based bit-serial (SFQ-BSE), bit-slice (SFQ-BSL), and bit-parallel processors (SFQ-BP). We also prepare the CMOS-based bit-parallel processor (CMOS-BP) as a comparison. Here, CMOS-BP refers to the configuration of a commercial processor that adopts the out-of-order instruction issuance scheme.

The delay parameters are summarized in Table 3.1. We use two delay parameters for the SFQ processor: the 1.0 $\mu$m Nb process currently used in SFQ circuits' demonstrations [57], and the 0.3 $\mu$m Nb process [40], where the scaling rule that holds for SFQ devices reaches its limit. We estimate the delay parameter of the 0.3 $\mu$m Nb process based on the scaling rule; when the JJ scales to $1/a$, the switching speed and delay are also $1/a$ [40]. $t_o$ is calculated by the sum of the arithmetic average of SetupTime and HoldTime of typical logic gates used in SFQ processors and the operating margin for countermeasures against manufacturing variations and jitter. $t_p$ is determined based on the CORE $1\beta$ [77, 87], that is one of state of the art SFQ processors. Specifically, $t_p$ is determined by multiplying the circuit delay of the SFQ adder that constitutes the EX stage, which is the critical path of CORE $1\beta$, by 7, which is the number of pipeline stages of CORE $1\beta$. The circuit delay of this SFQ adder can be obtained by multiplying the GDP period by the number of cycles required for processing by the SFQ adder, as shown in Eq. (3.2).

$$D_{SFQadder} = T_{GDP} \times (N_{stages} + N_{slices} - 1) \tag{3.2}$$

Where $D_{SFQadder}$ is the circuit delay of this SFQ adder, $T_{GDP}$ is the GDP cycle time for driving SFQ adder's gates, $N_{stages}$ is the number of pipeline stages of SFQ adder, and $N_{slices}$ is the number of slices. For example, $N_{slices}$ is 1 in SFQ-BP; on the other hand, $N_{slices}$ is the data bit width in SFQ-BSE because the data is processed bit by bit. We use the value of the SFQ cell library of the 1.0 $\mu$m Nb process as the circuit delay parameters. In addition, all circuit delays are assumed to be obtained from 64-bit SFQ adders. Below, we explain how to obtain $t_p$ for each of SFQ-BP, SFQ-BSE, and BSL.

In SFQ-BP, since there is no feedback loop in the SFQ adder, the GDP period $T_{GDP}$ is $t_o$, which is the maximum value that can be supplied.

In SFQ-BSE and SFQ-BSL, there is a feedback loop path in the SFQ adder, and it is necessary to wait for the input data in this loop, i.e., SFQ-BSE and SFQ-BSL cannot apply the clock skewing and the $T_{GDP}$ is determined by the critical path delay of the SFQ adder. Fig. 3.1 shows the gate-level circuit diagram of the SFQ adder. There are two candidates for the critical path, path (A) and path (B). In this adder, path (A) is longer than path (B), and the logic gate elements contained in path (A) are the AND gate, the confluence buffers (CB), and the

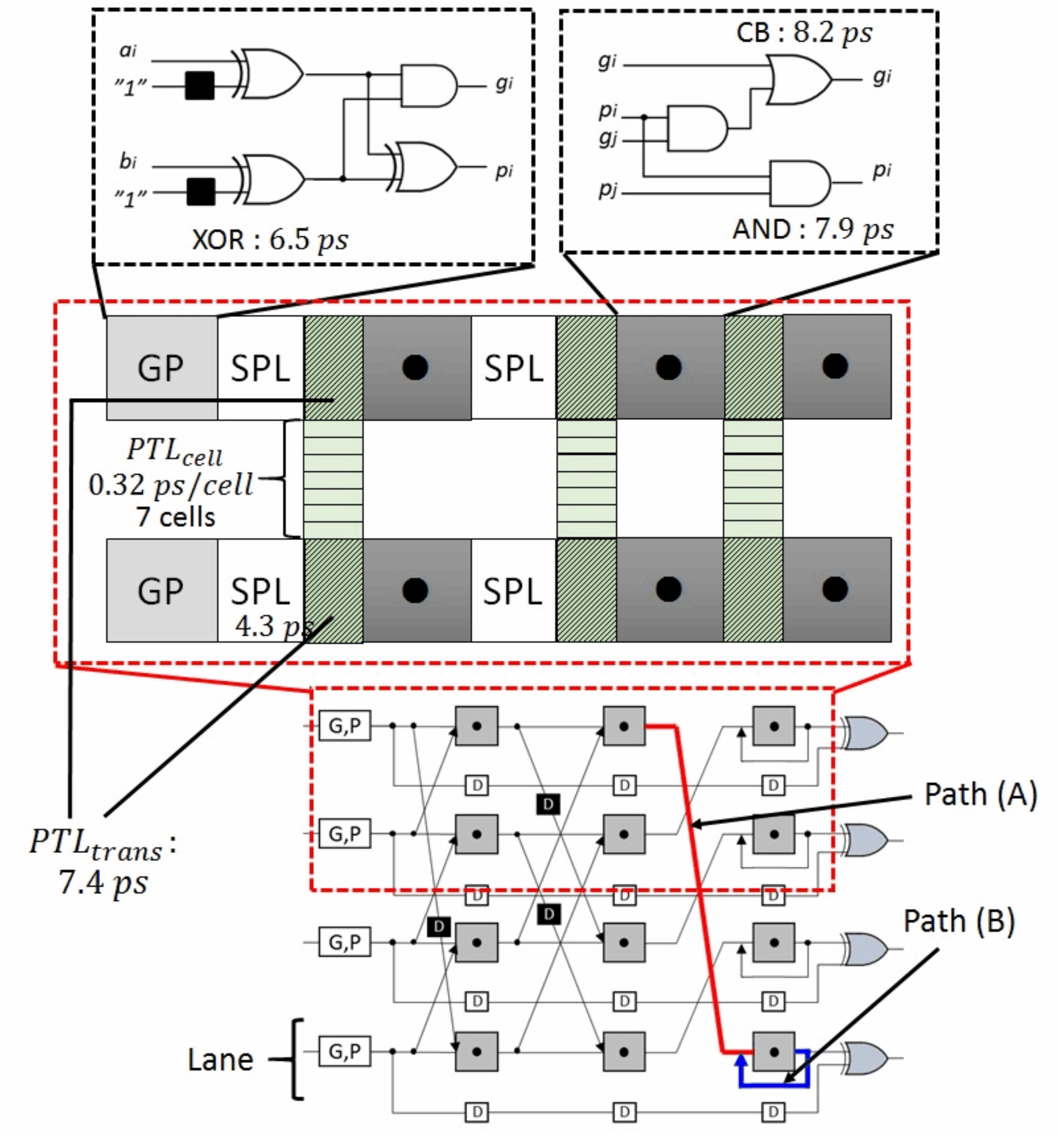


Figure 3.1: Gate-level circuit diagram of the SFQ adder.

superconducting passive transmission line (PTL), and splitters (SPL). The signal transmission time in PTL is determined by the delay required for transmission and reception to the transmission line ($D_{PTLtrans}$) and the transmission line's length. The delay of path (A) is calculated by Eq. (3.3).

$$\begin{aligned} D_{pathA} &= D_{AND} + D_{CB} + D_{PTLtrans} + D_{SPL} \\ &\quad + (DPW - 1) \times D_{PTLcell} \times N_{PTLcell} \end{aligned} \tag{3.3}$$

Where $D_{pathA}$ is the delay of path (A), $D_{AND}$, $D_{CB}$, $D_{SPL}$, and $D_{PTLcell}$ is the delay of AND gate, the confluence buffer, the splitter, and the PTL cell, respectively. $DPW$ is the data path bit width and $N_{PTLcell}$ is the number of PTL cells that exist between two bit lines as shown in Fig. 3.1. SFQ-BSL uses the 8-bit slice, which was the best-performing slice width for 64-bit data word lengths. The $t_p$ of bit-serial and slice processing in Table 3.1 are not just 64 times or 8 times that of the bit-parallel scheme, respectively. This is because the slice- and bit-level overlap execution reduces the processing delay. In the CMOS processor, referring to the configuration of Intel's 45 nm process Core i7 920, 14 stages of pipeline and clock frequency of 2.66 GHz are used for each parameter calculation [3]. $t_o$ and $t_p$ are calculated by the clock cycle time with the best ratio between the latch overhead and the delay of the pipeline stage (i.e., 1.8 to 6 [31]), as shown in Eq. (3.4), Eq. (3.5), respectively.

$$t_o = 1/2.66(\text{GHz}) \times 1.8/(1.8 + 6) \tag{3.4}$$

$$t_p = 1/2.66(\text{GHz}) \times p - t_o \times p \tag{3.5}$$

## 3.3 Design space exploration

This section evaluates each architecture parameter mentioned in Section 3.2.1 to determine the design guidelines for high-performance SFQ processors.

### 3.3.1 Datapath bit width evaluation

First, to show the effect of the datapath bit width on the processor performance, we evaluate the performance with the fixed number of pipeline stages. In this

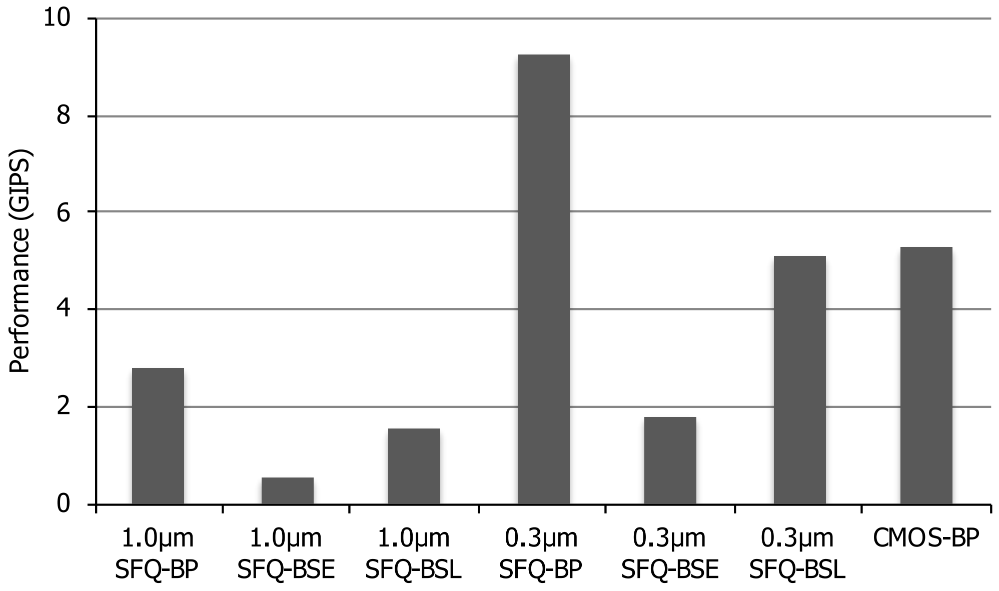


Figure 3.2: Performance comparison with fixed pipelines

evaluation, we set $\gamma = 0$ of the TPI shown in Eq. (3.1) to compare each processor model's peak performance. We use the number of instructions executed per second (IPS) as the performance metric. IPS is calculated by Eq. (4.2).

$$IPS = 1/TPI \tag{3.6}$$

Fig. 3.2 shows the IPS comparison when the number of pipeline stages is fixed (SFQ and CMOS processors consist 7 and 14 stages, respectively). As the comparison results, the performance of SFQ-BP, SFQ-BSE, and SFQ-BSL with 1.0 $\mu$m Nb process are 2.78 Giga IPS (GIPS), 0.52 GIPS, and 1.53 GIPS, respectively. In the 0.3 $\mu$m Nb process, these clock frequencies are 9.27 GIPS, 1.76 GIPS, 5.11 GIPS, respectively. In the bit-serial and bit-slice processing schemes, logic gates in the pipeline stage are activated multiple times during the processing of one-word data, resulting in increasing $t_p$. The increase of $t_p$ has a significant adverse effect on the IPS. The results clearly show that the bit-level parallelism of the bit-parallel processing scheme is more effective in improving the IPS than the latency reduction by the bit-level and slice-level overlapping execution.

In the 1.0 $\mu$m Nb process SFQ processors, the IPS of SFQ-BP, SFQ-BSE, and SFQ-BSL are about the same as or lower than that of CMOS-BP. In the

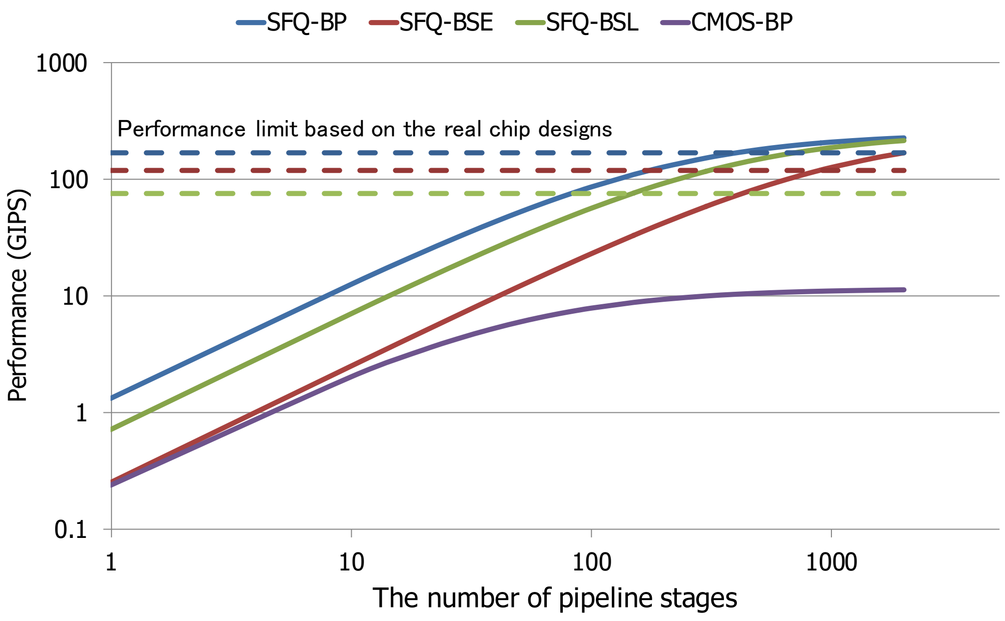


Figure 3.3: Performance comparison with variable pipeline stages

0.3 $\mu$m Nb process, the IPS of SFQ-BSE and SFQ-BSL are also about the same as or lower than that of CMOS-BP. On the other hand, SFQ-BSL and SFQ-BP achieve 5.11 GIPS and 9.27 GIPS, which are higher than CMOS-BP performance, respectively. This result also indicates that the bit-parallel processing scheme is the most suitable and sufficient for achieving high-performance SFQ processors.

### 3.3.2 Pipeline depth evaluation

Next, we evaluate the peak performance ($\gamma = 0$) with variable pipeline stages to show the effect of the pipeline depth. Fig. 3.3 shows the performance comparison between SFQ-BP, SFQ-BSE, SFQ-BSL with a 0.3 $\mu$m Nb process, and CMOS-BP. The y-axis represents the performance, i.e., IPS, and the x-axis represents the number of pipeline stages. According to Fig. 3.3, the SFQ processors have more capacity to improve their performance than CMOS processors by increasing the number of pipeline stages. This is because the $t_o$ of the SFQ circuit is much smaller than that of the CMOS circuit, as shown in Table 3.1. Therefore, SFQ processors can achieve higher clock frequency, i.e., higher performance. Moreover, in the CMOS processors, it is tough to improve the clock frequency by deepening

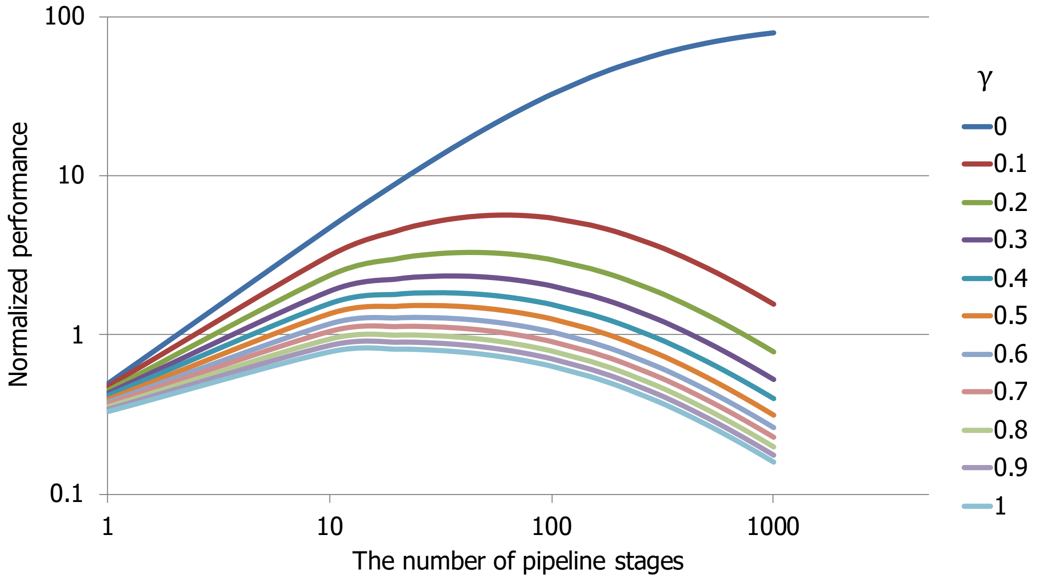


Figure 3.4: Performance comparison between SFQ-BP and CMOS-BP considering the ratio of the average pipeline stall $\gamma$

the pipeline due to their power consumption, i.e., the power-wall problem. In contrast, in the SFQ circuits, the power consumption problem does not occur due to their low-power nature.

The broken lines in Fig. 3.3 are the limit of peak performance calculated with the limit of clock frequency based on the real chip design data of SFQ-BP, SFQ-BSE, SFQ-BSL, which are 166.67 GIPS, 76.66 GIPS, and 119.90 GIPS, respectively. When the number of SFQ-BP's pipeline stages increases, the performance of SFQ-BP reaches 166.67 GIPS at 377 pipeline stages, which is higher than that of SFQ-BSE and SFQ-BSL with the same number of pipeline stages. On the other hand, CMOS-BP with an increased number of pipeline stages cannot outperform around 11 GIPS. To obtain higher performance in SFQ processors, SFQ processors should adopt a bit-parallel processing scheme and a deep pipeline structure. However, the effect of pipeline stall increases in proportion to the number of pipeline stages, increasing TPI, as shown in Eq. (3.1).

Next, let us consider the performance when pipeline stalls occur. Fig. 3.4 shows the performance comparison between SFQ-BP with 0.3 $\mu$m Nb process and CMOS-BP in case of $\frac{N_H}{N_I} = 0.5$ considering the ratio of the average pipeline stall, $\gamma$.

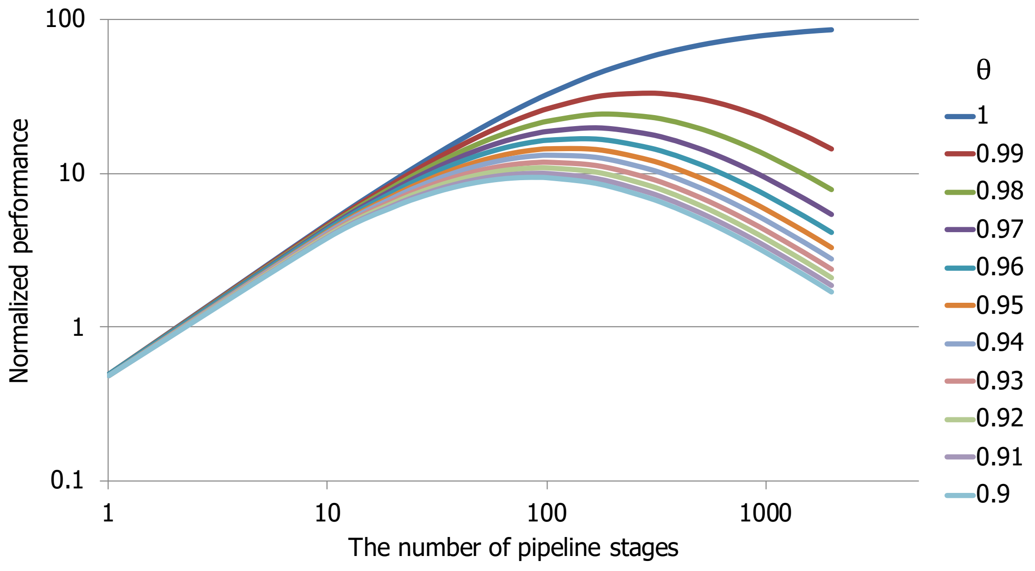


Figure 3.5: Performance comparison between SFQ-BP and CMOS-BP considering the pipeline stall concealment rate $\theta$

The performance is obtained by normalizing the IPS with the peak performance of CMOS-BP. When $\gamma = 0$, that is, there is no pipeline stall, SFQ-BP achieves 62.66 times the performance of the CMOM-BP. However, the performance ratio between degrades up to 5.67 in 60 pipeline stages when the pipeline stalls are considered ($\gamma = 0.1$). In other words, if the pipeline stalls cannot be sufficiently concealed, the performance will be low even with the deep pipeline structure. Thus, we introduce $\theta$, the pipeline stall concealment rate, to Eq. (3.1). $\theta$ can take a value from 0 to 1, and $\theta = 1$ indicates that all stalls are concealed. TPI with $\theta$ is calculated as shown in Eq. (3.7).

$$\begin{aligned} TPI = \frac{T}{N_I} &= (\frac{t_o}{\alpha} + (\gamma \times (1-\theta))\frac{N_H}{N_I}t_p) \\ &+ \frac{t_p}{\alpha p} + (\gamma \times (1-\theta))\frac{N_H}{N_I}t_o p \end{aligned} \tag{3.7}$$

Fig. 3.5 shows the performance comparison result in $\frac{N_H}{N_I} = 0.5$ and $\gamma = 0.5$. The y-axis represents the normalized performance, and the x-axis represents the number of pipeline stages. Fig. 3.5 shows the result of increasing the value of $\theta$ by 0.01 from 0.9 to 1, that is, increasing the pipeline stall concealment rate by 1% from 90% to 100%. In case that 99% of pipeline stalls are concealed, SFQ-BP

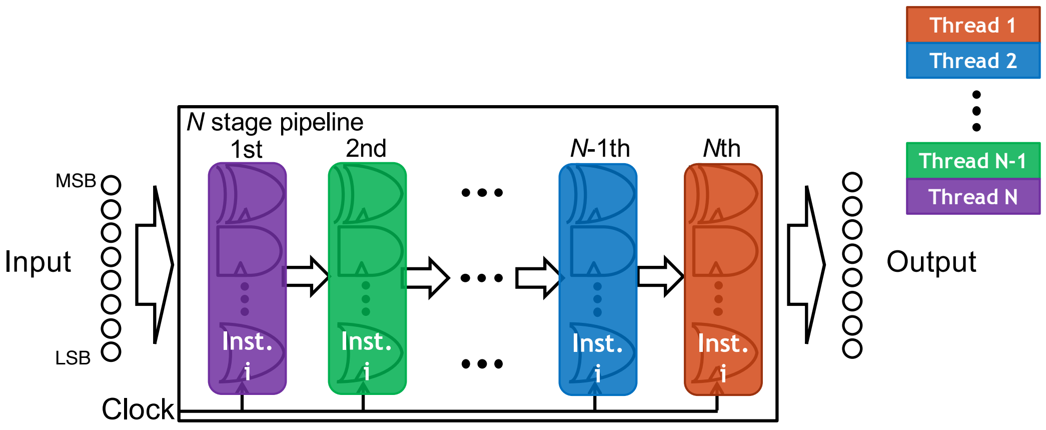


Figure 3.6: Overview of the proposed design policy

with 300 pipeline stages outperforms 32.98 times than CMOS-BP. As a result, by hiding 99% of the pipeline stalls, SFQ processors' performance has a capacity for improvement in an extremely deep pipeline structure, such as 300 pipeline stages. Besides, advanced pipeline stall concealment technology is essential for high-performance SFQ processors.

## 3.4 Design guidelines for high-performance SFQ processors

To realize a dramatic performance improvement by the SFQ processors, architects should introduce a microarchitecture that considers device and circuit characteristics and various design constraints. Therefore, this dissertation proposes the following policy for microarchitecture design, considering SFQ circuits and design technology's current situation and the result of architectural design space exploration. Fig. 3.6 summarizes the following design policy.

- **Bit-parallel processing scheme**: Compared to the bit-serial and bit-slice processing schemes that expand processing in the time direction, the bit-parallel processing scheme has the following three advantages. 1) As shown in Section 2.3.3, the bit-parallel processing scheme can avoid the feedback loop inside the combinational circuit that exists in the bit-serial and bit-slice processing schemes. As a result, the bit-parallel processing has the potential to achieve a higher clock frequency than that of other processing schemes. 2) The latency of combinational circuits can be reduced by exploiting bit-level parallelism. 3) Since iterative processing in the time direction is not required in the bit-parallel processing scheme, the timing adjustment can be relatively easy in circuit design and layout design.

- **Gate-level pipeline structure**: As shown in Section 3.3.2, SFQ processor can significantly benefit from increasing the number of pipeline stages due to its low-latency SetupTime and HoldTime. Gate-level pipelining is the most fine-grained pipeline structure, in which the clock cycle time depends on the delay of only one logic gate. Such super-pipelining cannot benefit from increasing the frequency in CMOS because of two primary reasons: the heat caused by increasing frequency, and another is frequency becomes limited by SetupTime and HoldTime of pipeline registers. However, these disadvantages do not appear in SFQ designs because SFQ circuits consume very little energy compared to CMOS technology, and SFQ logic gates take only a few ps to switch. Moreover, there is no overhead of additional pipeline registers due to their latch or memory functionality. Furthermore, this pipeline structure makes it possible to unify the GDP and global clock signal required for controlling the pipeline registers. In other words, the gate-level-pipelined processor can control the operation of the entire pipeline with the GDP, and it also has advantages from the viewpoint of design simplification. From now on, GDP is called the "clock" in this dissertation due to the gate-level pipeline structure.

- **Fine-grained multithread execution**: To achieve high performance by gate-level pipelining, SFQ processors must conceal almost all pipeline stalls. Because SFQ logic uses a combination of SFQ pulses for logic operations, adjusting the timing of pulses is one of the critical issues when targeting the frequency around 100 GHz, which makes it hard to implement conventional stall concealment technologies such as forwarding and branch prediction. Therefore, we use fine-grained multithreading with as many threads as the number of pipeline stages that interleaves those threads to execute an instruction every cycle. It can prevent pipeline stalls due to data and control hazards without increasing hardware complexity, i.e., no frequency degradation is occurred.

## 3.5 Conclusions

This chapter clarifies conventional SFQ processors' problems and quantitively shows the design guidelines for high-performance SFQ processors by conducting architectural design space exploration. Specifically, we focus on two key architectural parameters; datapath bit width and pipeline depth. As a result, we show the bit-parallel processing with gate-level pipelining is suitable for high-performance SFQ processors. To achieve high performance by such an ultra-deep pipelining, SFQ processors must conceal almost all pipeline stalls. For example, the performance improvement degrades 63 times to 5.7 times even with the small pipeline stalls are considered (i.e., $\gamma = 0.1$). Therefore, we propose fine-grained multithreading execution as the pipeline stall concealment technologies. The concept of the fine-grained multithreading execution is filling the pipeline with independent instructions. Thus, it can prevent all pipeline stalls caused by data and control hazards. In the next step, we should evaluate the feasibility and potential of the proposed architecture design guidelines.

# Chapter 4

# Prototype design of SFQ processor

## 4.1 Introduction

Due to SFQ device's ultra-high-speed and ultra-low-power natures, researchers have so far significantly contributed to developing SFQ devices and design technologies, and SFQ processors have been successfully demonstrated [87, 77, 64]. Although the SFQ processors operate with outstanding clock frequency, e.g., several dozen GHz or even more than 100 GHz, unfortunately, their effective performance regarding "program execution speed" is comparable or worse than that of state-of-the-art CMOS processors. The fundamental problem existing behind the SFQ processors is the lack of optimization from the viewpoint of microarchitecture.

Therefore, in Chapter 3, we have explored the architectural design space for SFQ processors by standing on a device/circuit/architecture level co-designs. As the architecture exploration results, we have proposed bit-parallel processing and gate-level-pipelining with fine-grained multithreading for achieving extremely high performance. We have designed an 8-bit bit-parallel, gate-level-pipelined ALU and have successfully demonstrated operations of over 56 GHz with 1.6 mW. However, it is still not clear how much the gate-level pipelined and bit-parallel organization can improve the performance and power efficiency at the whole processor level. Moreover, because SFQ circuits need a cryocooler that keeps the circuits in 4 kelvin, it is necessary to evaluate the cryocooler's effect on power consumption.

To solve these problems, we have designed and fabricated a 4-bit SFQ processor chip as a prototype. Moreover, we verify its operation by chip measurement. As a result, the prototype chip has successfully operated at 32 GHz with 6.5 mW. Besides, the resultant power efficiency achieves 2.5 tera-operations per watt (TOPS/W). Based on these results, we have estimated the 64-bit processor and evaluated the power efficiency, including a cryocooler cost for 4 kelvin. The power efficiency with the cooling cost is estimated to at most 7.1 GOPS/W.

## 4.2 Specification of prototype processor

### 4.2.1 Architectural design guidelines

The architecture which considers device features and design limitations are required to realize ultra-high-performance SFQ processors. We have explored the architectural design space of SFQ processors in Chapter 3. As a result, we have reached the following conclusions.

- **Bit-parallel processing**: Unlike bit-serial or bit-slice operations, the bit-parallel processing handles the processor's word size at the same time in parallel. The latency of word-size operations can be reduced by exploiting bit-level parallelism.

- **Gate-level pipelining**: Gate-level pipelining is the most fine-grained pipeline structure, which the clock cycle time depends on the delay of only one logic gate. Because SFQ logic gates have a latch function and take only a few picoseconds to switch, the pipeline structure can benefit from increasing the SFQ processors' frequency without the overhead of inserting additional pipeline registers.

- **Fine-grained multithreading**: To achieve high performance by introducing gate-level pipelining, almost all pipeline stalls caused by data and control hazards must be concealed. However, it hard to implement conventional stall concealment technologies such as forwarding and branch prediction because such conventional schemes can incur significant frequency degradation in the gate-level pipeline structure. Therefore, it is essential to hide almost all pipeline stalls without increasing hardware complexity. For this challenge, we apply a fine-grained multithreading execution model with as many threads as the number of pipeline stages. It can prevent pipeline stalls due to data and control hazards without any frequency degradation.

### 4.2.2 Instruction execution scheme

As described in Section 4.2.1, a large-scale fine-grained multithreading execution is adopted. Initially, it is necessary to maintain a program counter for each thread and switch the execution thread every clock cycle. However, it is challenging to prepare a memory that can follow a processor operating at several dozens of GHz and read instructions every cycle. Therefore, we employ a single instruction multiple threads (SIMT) execution. In the SIMT execution, the execution thread is switched every clock cycle, but all threads execute the same instruction. By expanding a single instruction in the time direction, the instructions' reading time is reduced in proportion to the number of threads for the operation of the processor. Thus, the latency gap between the processor and the memory can be hidden.

Besides, to realize large-scale fine-grained multithreading, we introduce a register file using a circulated shift register (from now on referred to as a circular register file). It is necessary to switch the architecture state following to processor's speed. Moreover, the most practical SFQ memory technology that can operate at the same speed as a processor is the shift register memory.

If one entry of the shift register corresponds to the register set for one thread, the register set at the access port can be replaced every cycle. Moreover, it performs a non-destructive readout due to its circular structure. The circuit scale of the register file will increase in proportion to the number of threads. The number of threads is reduced to half the number of pipeline stages to keep it at a level feasible in the current fabrication process. In this case, it is necessary to schedule the instructions so that there is no dependency between two consecutive instructions.

Table 4.1: Instruction set

| instruction | Operation | Opcode | Instruction format |
|---|---|---|---|
| ADD | Integer addition | 100000 | Binary operation instruction |
| SUB | Integer subtraction | 101000 | Binary operation instruction |
| ADDS0 | Conditional integer addition | 100100 | Binary operation instruction |
| SUBS0 | Conditional integer subtraction | 101100 | Binary operation instruction |
| ADDI | Integer immediate addition | 110000 | Binary operation instruction |
| SUBI | Integer immediate subtraction | 111000 | Binary operation instruction |
| LW | Load word | 111100 | Data transfer instruction |
| LI | Load immediate | 010000 | Data transfer instruction |
| SW | Store word | 010100 | Data transfer instruction |
| SK6S0 | Conditional skip instruction | 000010 | Control instruction |
| HLT | Halt | 000001 | Control instruction |
| NOP | Non operation | 000000 | Control instruction |

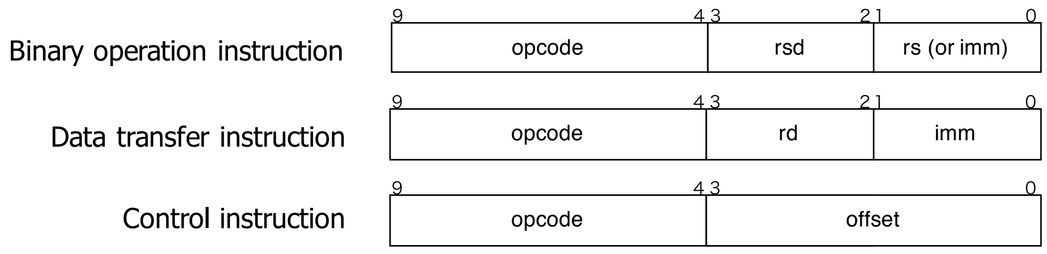


Figure 4.1: Instruction format

### 4.2.3 Instruction set

Our processor employs a unique reduced instruction set computer (RISC) based instruction set considering SFQ devices' characteristics and current design constraints. Table 4.1 shows the implemented instructions. The bit width of instructions is fixed at 10 bits, and the instruction format is shown in Fig. 4.1. The upper 6-bits instruction represents the operation code (opcode). *rs* is the source register's address for the operation, and *rd* is the destination register's address. *rsd* indicates the address of a register that is both a source register and a destination register, and it is used only for binary operation instructions. *imm* is 2-bit immediate data embedded in the instruction, and it is expanded to 4 bits and used as an operand for arithmetic operations and as an address for data transfer instructions.

In addition to the basic operation instructions and data transfer instructions, we add some conditional operation and control instructions that consider the processor's instruction execution and SFQ design constraints. In the SIMT execution, because the same instruction is executed in all threads, some mechanism is necessary for performing different processing in each thread. Therefore, we add some conditional operation instructions. Specifically, the processor holds a flag bit for each thread, referring to the flag bit when executing a conditional instruction and changing the processing accordingly.

Moreover, we add a control instruction that assumes an instruction memory consisting of a 20-entry circular register file. A more complex instruction sequence can be executed with a limited number of entries by controlling the number of circulations and skipping instructions. Specifically, the instruction memory is divided into three areas: the initialization area, the kernel area, and the termination processing area. The instruction in the initialization area is executed only at the beginning of the program sequence. The kernel area is executed a specified number of times, and the instruction ends after the specified number of executions. A conditional skip instruction is employed to execute instructions in the termination processing area after the kernel area execution.

### 4.2.4 Test program

One of the design objectives of the SFQ processor based on the architectural design guidelines is to demonstrate an effective program. An "effective program" is not just a list of instructions but a somewhat complicated program that processes data for some purpose. It is necessary to consider the program that has an advantage in SFQ processors and can be executed with limited hardware resources. Specifically, it is necessary to consider that the instruction memory of the SFQ processor consists of a circular shift register, and the number of entries (that is, the number of instructions that can be stored) is 20. Furthermore, since the processor employs SIMT execution, it seems suitable for applications that perform the same processing on a large amount of data. The matrix-vector product is one of the basic programs that satisfy these conditions. The matrix-vector product is used in various fields such as chemical calculations and neutral network calculations, and

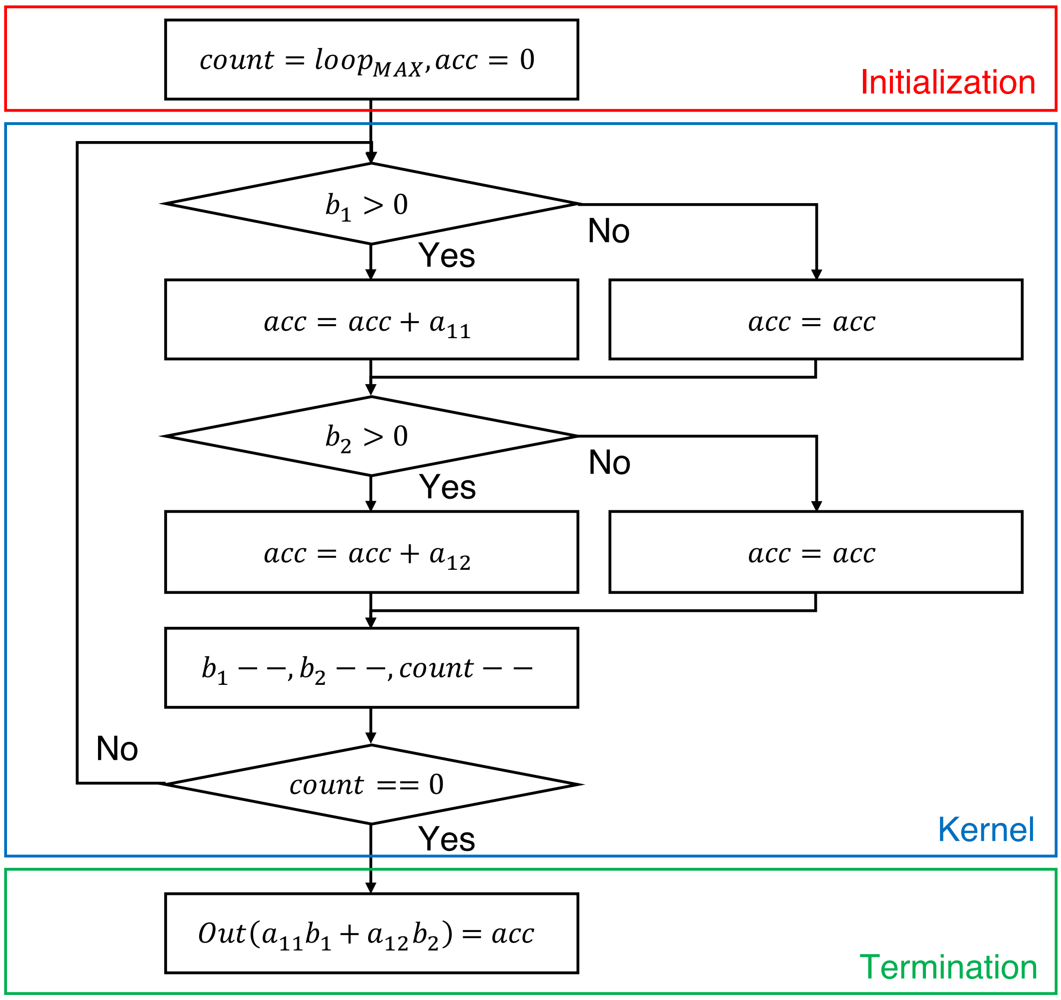


Figure 4.2: Algorithm of the 2-by-2 matrix-vector product

the calculations of matrix elements are independent of each other and are highly compatible with multithreaded execution.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_{11}b_1 + a_{12}b_2 \\ a_{21}b_1 + a_{22}b_2 \end{pmatrix} \tag{4.1}$$

Therefore, we use a program that calculates one element of a 2-by-2 matrix and a 2-dimensional column vector, as shown in Eq. (4.1), as a test program. Since the processor does not implement the multiplication instruction, the product is performed by repeating the addition. The basic algorithm is shown in Fig. 4.2. *acc* is the accumulator that adds the calculation results, and *count* is the counter that represents the number of iterations required for the accumulation. Fig. 4.2

Table 4.2: Assembly code of the 2-by-2 matrix-vector product

| | | | | |
|---|---|---|---|---|
| 0: | LI | $r2$ | 0x1 | # Set *count* to 1 |
| 1: | LI | $r3$ | 0x0 | # Set *acc* to 0 |
| 2: | NOP | | | |
| 3: | LW | $r1$ | $M[2]$ | # Load the multiplier |
| 4: | LW | $r0$ | $M[0]$ | # Load the multiplicand |
| 5: | SUBI | $r1$ | 0x1 | # Decrement the multiplier & set the value of the sign flag |
| 6: | ADDS0 | $r3$ | $r0$ | # Add if the value of the sign flag is 0 |
| 7: | SW | $r1$ | $M[2]$ | # Store the multiplier |
| 8: | LW | $r1$ | $M[3]$ | # Load the multiplier |
| 9: | LW | $r0$ | $M[1]$ | # Load the multiplicand |
| 10: | SUBI | $r1$ | 0x1 | # Decrement the multiplier & set the value of the sign flag |
| 11: | ADDS00 | $r3$ | $r0$ | # Add if the value of the sign flag is 0 |
| 12: | ST | $r1$ | $M[3]$ | # Store the multiplier |
| 13: | SUBI | $r2$ | 0x1 | # Decrement *count* & set the value of the sign flag |
| 14: | NOP | | | |
| 15: | NOP | | | |
| 16: | NOP | | | |
| 17: | NOP | | | |
| 18: | NOP | | | |
| 19: | SK6S0 | | | # Skip the initialization and termination area if the value of the sign flag is 0 |
| 20: | NOP | | | # Delay slot for skip instruction |
| 21: | SW | $r3$ | $M[0]$ | # Store the operation result |
| 22: | NOP | | | |
| 23: | HLT | | | # Termination |

assumes the case of calculating the elements $a_{11}b_1 + a_{12}b_2$ of Eq. (4.1). In the initialization area, *count* is set to $loop_{MAX}$, the maximum number of iterations in the kernel area, and *acc* is set to 0. In the kernel area, the multiplicand is added to *acc* according to the sign of the multiplier result. At the end of the kernel area, the multiplier and *count* are decremented, and *count* is determined whether 0 or not. If *count* is 0, the number of iterations in the kernel area reaches to $loop_{MAX}$, and the process moves to the termination area. Finally, the result of *acc* is output as $a_{11}b_1 + a_{12}b_2$ of Eq. (4.1).

The operation flow is shown in Fig. 4.3, and the assembly code is shown in Table 4.2. The assembly code is shown in Table 4.2. Considering the current design constraints, the number of general-purpose registers is 4 per thread, the number of sign flag registers used for conditional arithmetic instructions is 1 per thread, the number of instruction memory entries is 24, and the number of data

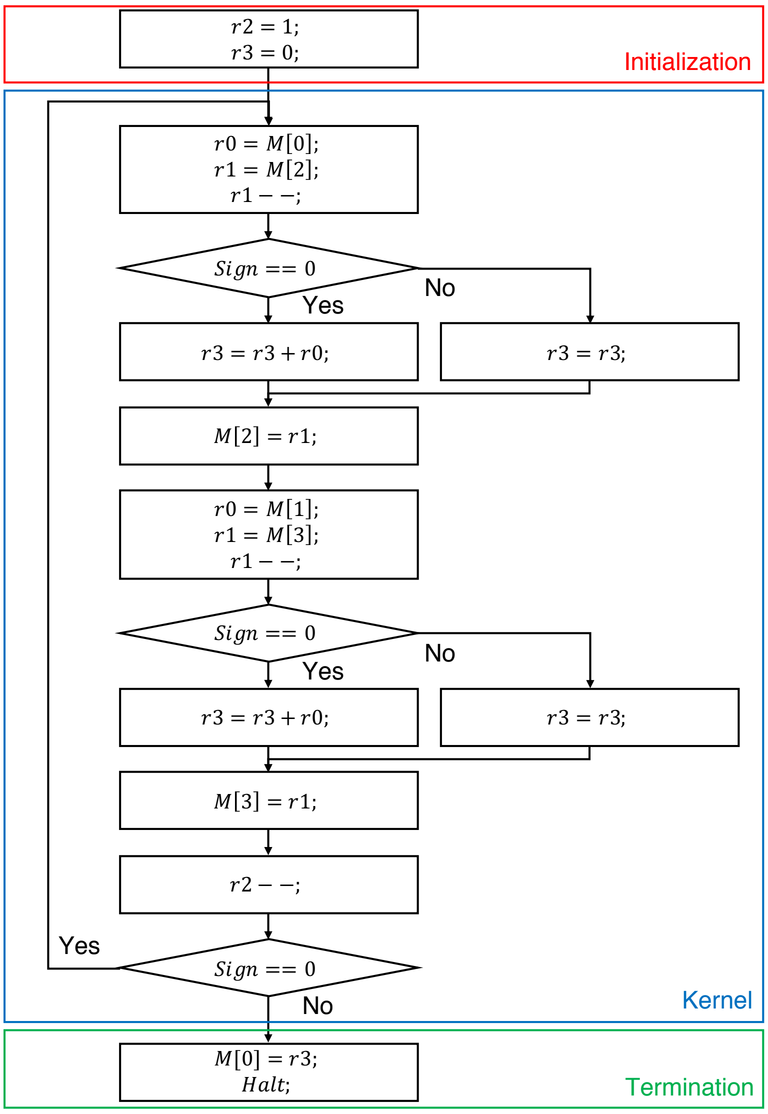


Figure 4.3: Operating flow chart of the 2-by-2 matrix-vector product

memory entries is 4 per thread. To avoid overflow, we set the matrix elements values from -4 to 3, and the vector elements values from 0 to 2. Since the vector element is a multiplier, the maximum number of loops, $loop_{MAX}$, is 2. $r0$ and $r1$ in Fig. 4.3 are registers that hold the multiplier and multiplier, respectively. $r2$ and $r3$ are registers corresponding to *count* and *acc* in Fig. 4.3, respectively.

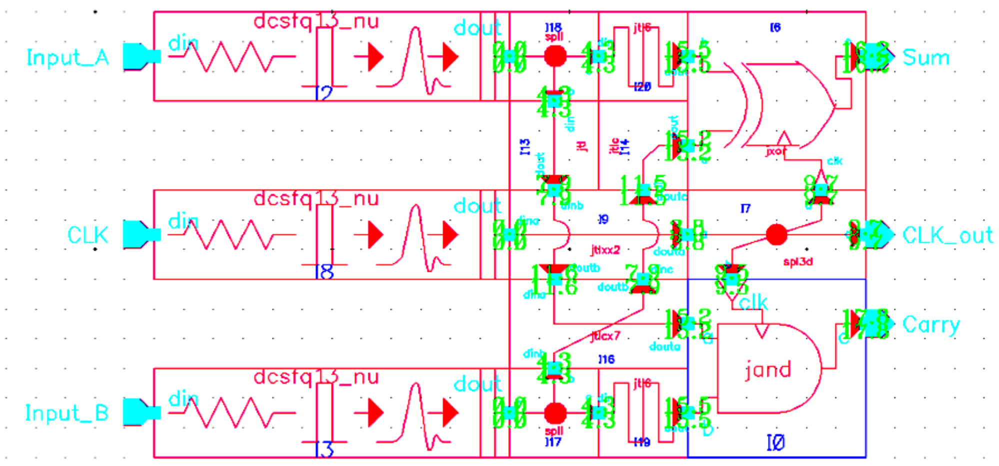


Figure 4.4: The layout of 1-bit half adder

## 4.3 Design and implementation

### 4.3.1 Design methodology

In this section, we design the processor by a method called *cell-based design* with the layout editor called Cadence "Virtuoso". It is a hierarchical design scheme to design the target circuits by combining the cells, which are the basic circuits such as logic gates or wiring designed in advance. We use a cell library developed in co-operation with the International Superconductivity Technology Institute (AIST), which is compatible with the advanced process (10kA/cm$^2$ process with 9 layers of niobium) [57]. Specifically, we design SFQ circuits by the custom layout method, i.e., we put SFQ cells manually. Fig. 4.4 shows the 1-bit half-adder layout based on SFQ circuits as an example. In this design method, the designer must put not only the logic gate cells but also the wiring cells to design entire circuits due to the lack of design automation tools. The designer can check the clock's arrival timing and data pulses by using a static timing analyzer (STA). The STA shows both the first and last signals' arrival, as shown in Fig. 4.4.

Fig. 4.5 shows the design flow of the prototype processor. First, we verify each module's correct operation at low speed (the clock cycle time is sufficiently large for the pulse transmission delay such as 1 to 5 GHz and does not incur timing errors). If an error is found, it is a logical error, not a timing error. After con-

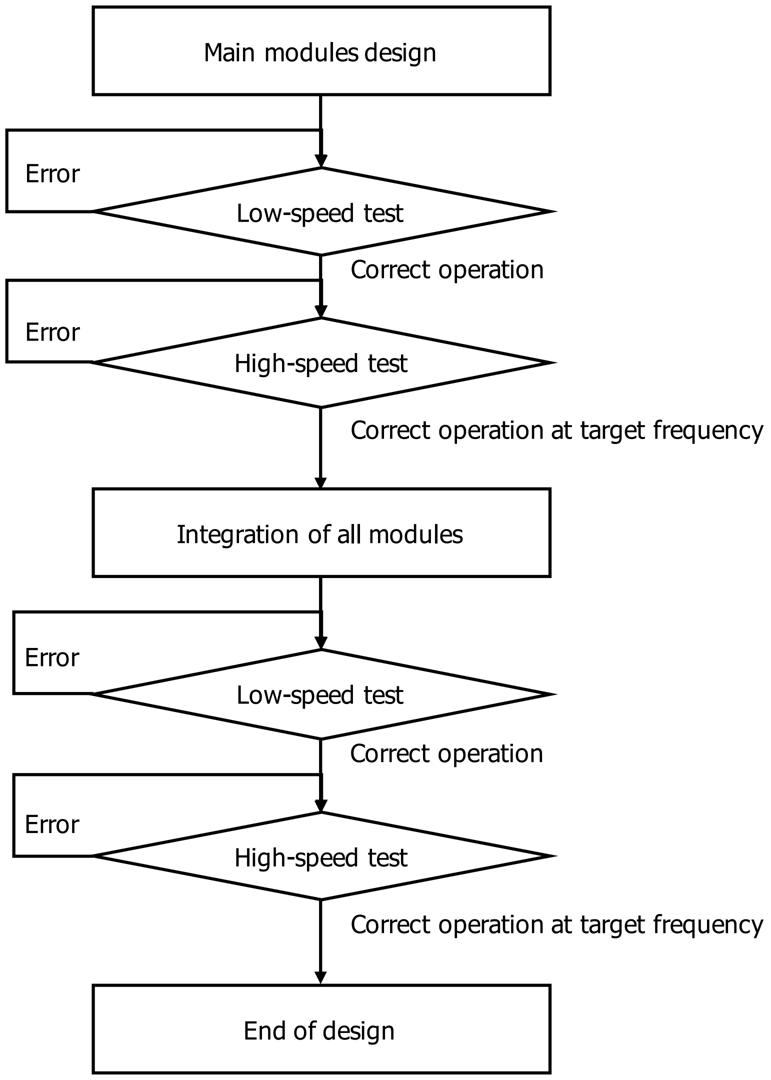


Figure 4.5: Design flow of the prototype processor

firming the correct operation at low speed for each module, we test the operation at high speed (i.e., several tens of GHz). If an error is found at a high-speed test, it is a timing error of SFQ pulses. The timing should be adjusted by static timing analysis and waveform observation of the simulation result. The timing adjustment is continuously performed until confirming the correct operation above the target clock frequency. After confirming each module's correct operation at both low speed and high speed, the modules are integrated into the processor. We verify the processor's operation in the same manner as each module test, and the design is completed when the correct operation is confirmed above the target clock

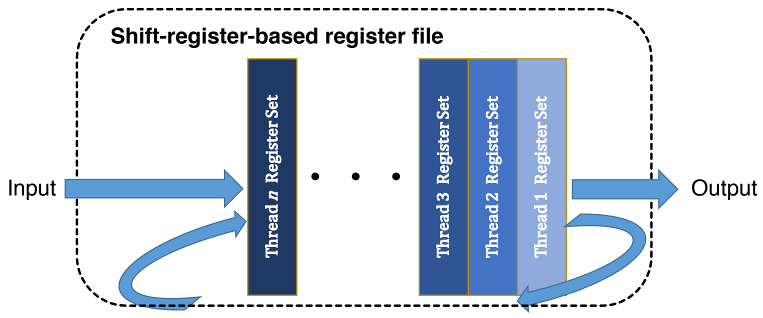


Figure 4.6: Shift-register-based register file for fine-grained multithreading

frequency.

We set the target frequency by following the past designs of SFQ processors. Specifically, we refer to CORE 1$\beta$, one of the typical SFQ processors designed so far [77, 87]. Although the clock frequency of CORE 1$\beta$ is 1.5 GHz, each logic gate operates at 25 GHz. This is because CORE 1$\beta$ a bit-serial processing scheme and a shallow pipeline structure, not the gate-level pipelining. On the other hand, our processor employs a bit-parallel processing and gate-level pipeline structure, and gates' operation speed represents the circuit clock frequency. Therefore we set a target frequency to 25 GHz.

### 4.3.2 Design challenges and solutions

There are two following challenges for designing the high-performance SFQ processor supporting the specification shown in Section 4.2; 1) how to realize the large register file for fine-grained multithreading, 2) more severe timing design compared to conventional SFQ processors. We resolve the above problems as follows.

- **Introducing the shift-register-based register file for fine-grained multithreading:** SFQ processor needs a large register file that can store all threads' architectural states and output an appropriate register set to the subsequent pipeline stage for each clock cycle. On the other hand, although several proposals about the SFQ on-chip memory technology have been made so far, the most practical is the shift register-based memory. With the consideration of the design constraint, we introduce the shift-register-based register file for fine-grained multithreading, as shown in Fig. 4.6. The input and out-

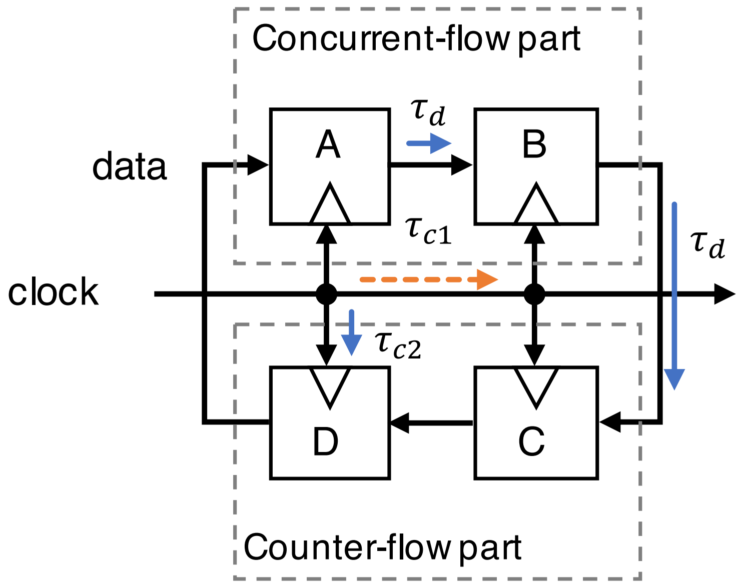


Figure 4.7: *Brahcn clocking*: combination of concurrent- and counter-flow clocking

put of the shift register are connected for realizing non-destructive access. Each entry of the shift register stores every thread's architectural states' value and the executed thread is switched every cycle.

- **Clock arrival timing optimization:** The main concept of the clocking scheme of SFQ logic is not synchronized with all the gates like conventional CMOS designs but point-to-point (or gate-to-gate) synchronization, as shown in Section 2.3.3. There are mainly two clocking schemes in SFQ logic designs; the concurrent-flow (Fig. 2.10(a)) and the counter-flow clocking (Fig. 2.10(b)). The former is used for simple feedforward logic such as multiplier [55], whereas the latter is suitable for feedback loops. Since our processor have both the parts of feedforward and feedback logics, we apply the combination of concurrent- and counter-flow clocking schemes, called *branch clocking*, as shown in Fig. 4.7. The branch clocking can reduce the wire length of the feedback loop. Moreover, if we apply the concurrent-flow part for relatively complex circuits and the counter-flow part for simple circuits, the circuits can utilize the advantage of concurrent-flow clocking for increasing clock frequency in complex circuits without frequency degradation from the concurrent-flow clocking part.

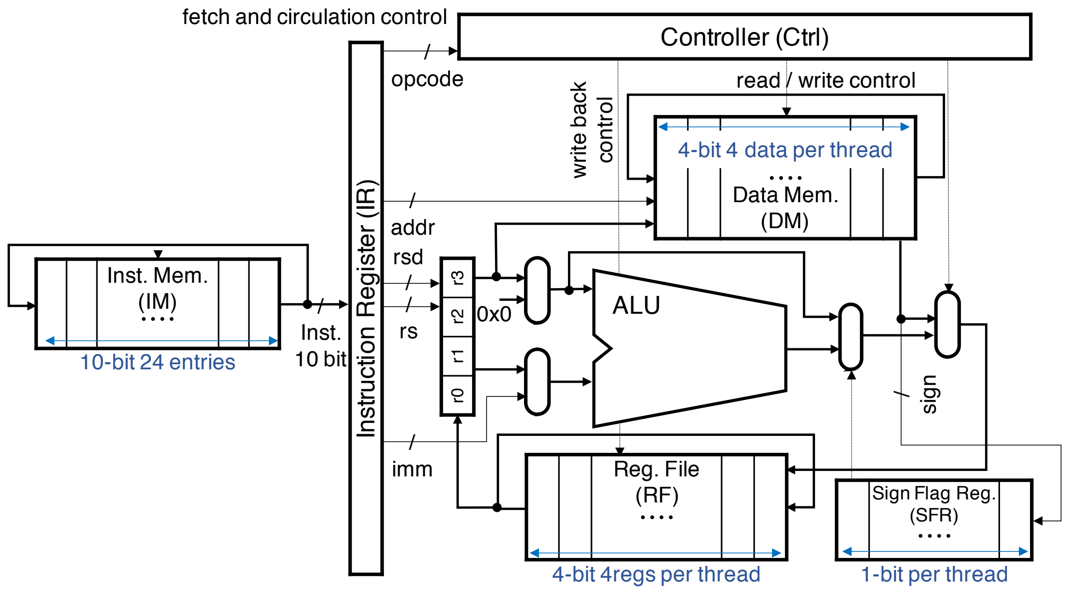


Figure 4.8: Microarchitecture of prototype processor

We use the combination clocking in the datapath with a large feedback path consisting of ALU and register files. Specifically, we apply concurrent-flow clocking in the ALU part and counter-flow clocking in the register file part. Therefore, in the ALU part, the concurrent-flow clocking can hide the data transfer delay, resulting in high-speed operation. On the other hand, in the register file part, the operation speed is still high even with the counter-flow clocking because the register file mainly consists of DFF gates, which is the fastest gate in SFQ design.

## 4.3.3 Microarchitecture

Fig. 4.8 shows the microarchitecture of the prototype processor. The processor consists of 24 pipeline stages. Taking the current SFQ integration technology into account, instructions and data are 10-bit and 4-bit wide, respectively. Besides, we employ SIMT execution to suppress the increase of a circuit scale, as shown in Section 4.2. We set the number of threads to 12 rather than 24 in this prototyping to satisfy a strict area constraint. Although such degradation makes the total area required to implement register file (RF) half, it also halves the peak performance because we need to reduce the instruction issue bandwidth to half to ensure the pipeline interlock-free streaming execution. The processor is composed of a 240-bit instruction memory (IM), a 16-bit data memory (DM) per each thread, a 10-bit instruction register (IR), a register file (RF) consists of 4-bit arithmetic registers (r0, r1, r2, r3) per each thread, a 1-bit sign flag register (SFR) per each thread, a 4bit-ALU, and a controller. As mentioned in Section 2.4.4, SFQ circuits cannot efficiently implement large-scale RAM because of its low driving capability. Considering this issue, memories of the processor consist of loop-shaped shift registers that input its output unless the values are not updated. Especially, RF consists of a shift register with entries as the same number as threads. Each entry has an architectural state associated with a thread and circulates by synchronizing with the SIMT execution.

## 4.4 Evaluations

In this section, first, we verify the correct operation of the prototype 4-bit SFQ processor chip. Next, we discuss the impact of a 64-bit SFQ processor by estimating its power consumption and clock frequency.

### 4.4.1 Verification results of 4-bit processor

The 4-bit SFQ processor chip is designed and implemented using the CONNECT cell library [88] and the AIST 10kA/cm$^2$ advanced 1.0 $\mu$m Nb process [57]. Fig. 4.9 shows the microphotograph of the 4-bit processor chip. It is composed of 23,713 JJs on 4.1 $\times$ 5.3mm$^2$ area.

We verify our processor in two ways: by post-layout simulation and by fabricated chip measurement. In the post-layout simulation, we use Cadence's Verilog-XL simulator. The netlist described in verilog of the processor is extracted from its layout, and the Verilog-XL run the simulation considering all wire and gate delay. We confirm the processor's correct operation up to 31 GHz with 6.9 mW in the test program shown in Section 4.2.4.

Fig. 4.10 shows the chip measurement setup. We use an electromagnetically shielded room to eliminate the influence of external magnetic fields as much as possible (Fig. 4.10(a)). Besides, we utilize the liquid helium bath for cooling chip

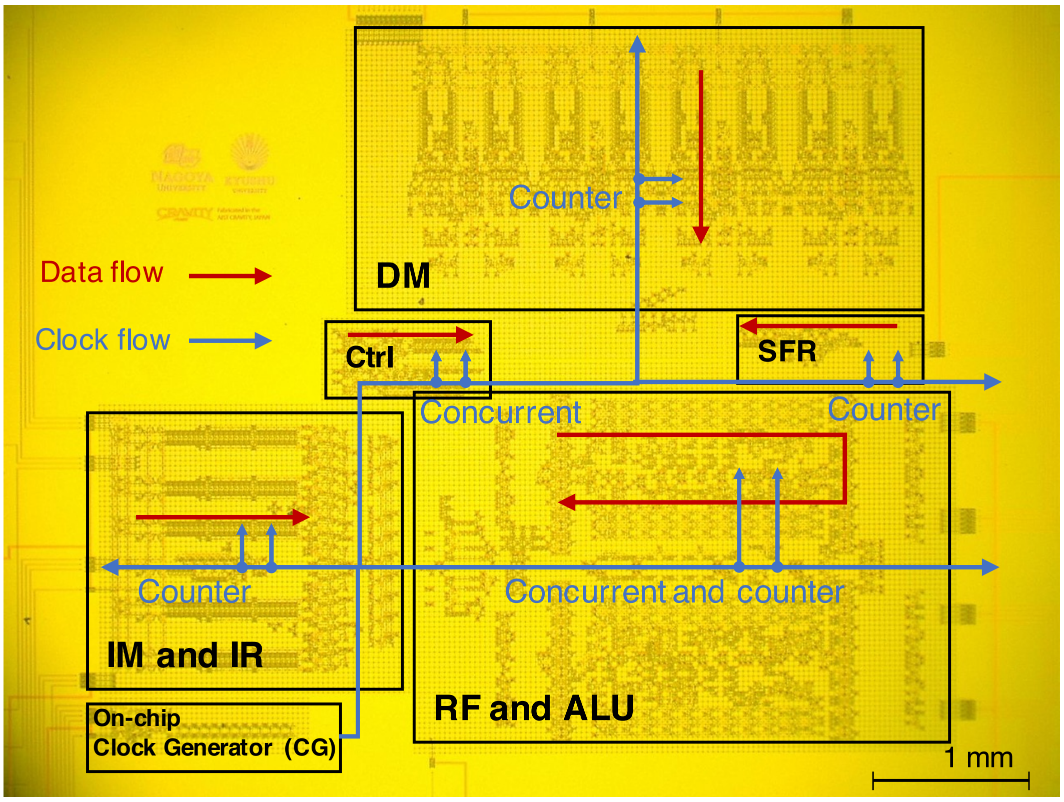


Figure 4.9: Microphotograph of the 4-bit processor chip

at 4 kelvin (Fig. 4.10(b)). Fig. 4.11 illustrates the on-chip high-speed testing methodology. First, we write instructions and initial data to on-chip memory at low speed by using an external data generator. Next, we input *trigger* signal, which triggers the on-chip clock generator, to start high-speed testing. After the end of testing, we check the final architectural states (i.e., register file and data memory) by using an external oscilloscope.

Fig. 4.12 displays the frequency dependence of operating margin in supply voltage. Y-axis shows the bias voltage, and x-axis shows the clock frequency. In the chip measurement, we confirm the correct operation of all instructions up to 32 GHz with 6.5 mW.

### 4.4.2 Evaluation of 4-bit processor

In this section, we evaluate the performance and power consumption of the 4-bit processor based on the verification results shown in Section 4.4.1. We use

(a) Electromagnetic shielded room

(b) Liquid helium bath

Figure 4.10: Chip measurement setup

operations per second (OPS) for performance metric which is given by Eq. (4.2),

$$\mathrm{OPS} = f \times \mathrm{OPC} \tag{4.2}$$

where $f$ and operations per cycle (OPC) is the clock frequency and the number of executed instructions per cycle, respectively. Because our design halves the number of register files compared to the number of pipeline stages to satisfy the chip area constraints, the instruction issue bandwidth (i.e., OPC) is 0.5. Therefore, the peak performance is 16 giga operations per second (GOPS). Moreover, the resultant power efficiency achieves 2.5 TOPS/W.

Fig. 4.13 shows the power consumption breakdown of the 4-bit processor. In the SFQ circuits, the static power consumption accounts for most of the power consumption, it largely depends on the circuit scale. In this design, the memory occupies most of the processor due to the SIMT execution, resulting in high memory power consumption.

### 4.4.3 Extension to 64-bit processor

In this section, we estimate clock frequency and the number of JJs of the 64-bit processor based on the evaluation results of the 4-bit processor and discuss the impact of the 64-bit SFQ processor, including the cooling cost for 4 kelvin. At first, we estimate the clock frequency of the 64-bit processor. There are two primary factors that affect the clock frequency when the bit width is extended: 1) the

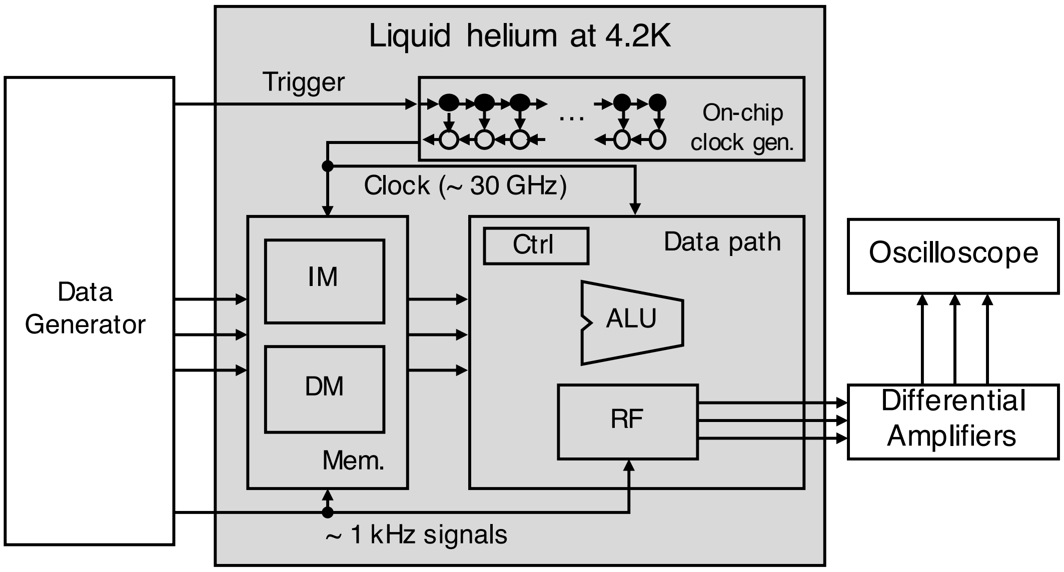


Figure 4.11: Illustration of on-chip high-speed testing

Table 4.3: The number of JJs of each 64-bit extended module

| Module name | $JJ_{64Reg}$ | $JJ_{64Dmem}$ | $JJ_{64ALU}$ | $JJ_{64MUX}$ | $JJ_{64Sign}$ | $JJ_{64Ctrl}$ | $JJ_{4ALU}$ |
|---|---|---|---|---|---|---|---|
| The number of JJs | 48,864 | 44,064 | 23,494 | 8,880 | 54 | 216 | 405 |

long wiring overhead, 2) the increase of jitter. For simplicity, the 64-bit processor is assumed to operate at 32 GHz, which is the same as that of the 4-bit design. Although there is the report about the impact of increasing jitter for frequency is not large [10], the scale of the 64-bit processor is quite larger than that of the 4-bit design, and it can incur the clock frequency degradation. The experimental study of the impact of extending the bit width for clock frequency is future work.

Secondly, we estimate the number of JJs of the 64-bit processor. The number of JJs of the 64-bit processor is given by Eq. (4.3),

$$JJ_{64MP} = JJ_{4MP} + JJ_{64EXT} \tag{4.3}$$

where, $JJ_{64MP}$ is the number of JJs of 64-bit processor, $JJ_{4MP}$ is the number of JJs of 4-bit processor, and $JJ_{64EXT}$ is the additonal JJs which needs to extend bit width from 4 to 64. $JJ_{64EXT}$ is given by Eq. (4.4)),

$$JJ_{64EXT} = JJ_{64EXT\_logic} \times \gamma \tag{4.4}$$

where, $JJ_{64EXT\_logic}$ is the number of JJs of additional logic gates, i.e., $JJ_{64EXT}$

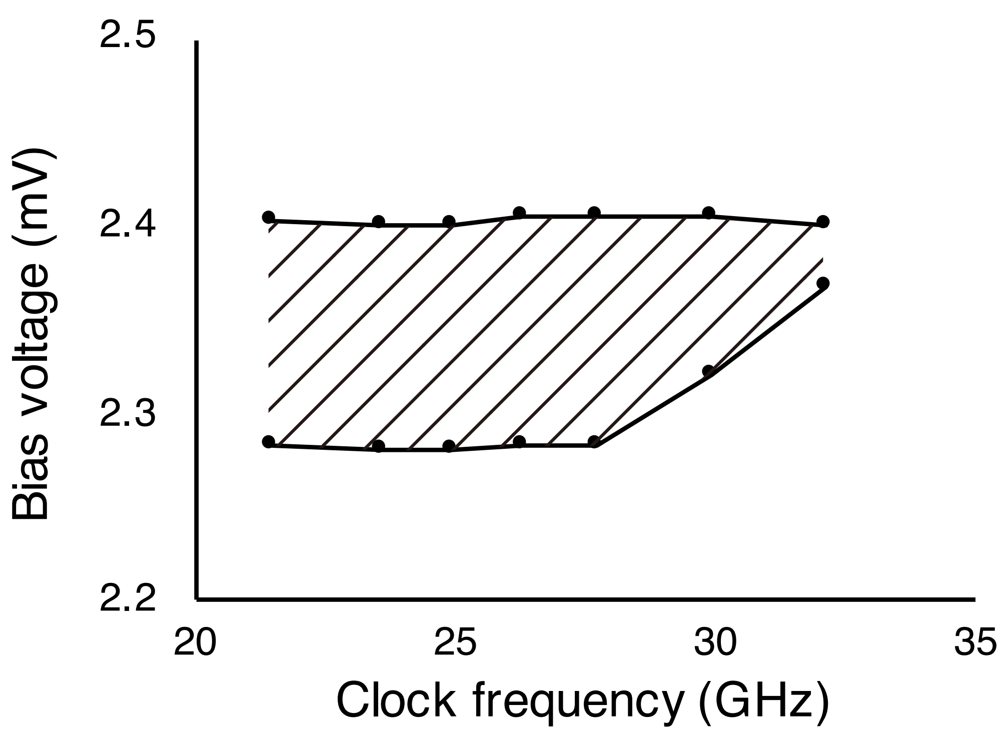


Figure 4.12: Frequency dependence of operating margin in supply voltage

without the cost of wiring, and $\gamma$ is the ratio of JJs of wiring to that of logic gates. $JJ_{64EXT\_logic}$ is calculated by the sum of additional JJs of component parts of processor such as ALU, multiplexer, register file, controller, and data memory. $JJ_{64EXT\_logic}$ is given by the sum of additional JJs of component parts of processor shown in Section 4.4.3.

$$JJ_{64EXT\_logic} = JJ_{64Reg} + JJ_{64Dmem} + JJ_{64MUX} + JJ_{64Ctrl} + JJ_{64ALU} \quad (4.5)$$

We model the number of JJs of component parts of processor such as ALU, multiplexer, register file, controller, and estimate the number of additional JJs of each component which shown in Table 4.3. We assume that the number of JJs of these components linearly increases with its bit width, and $JJ_{64EXT\_logic}$ is estimated to about 125,000 JJs. We use 2.08 as $\gamma$, which is calculated based on 4-bit processor configuration. According to Equation (4.4), $JJ_{64EXT}$ is calculated to about 260,000 JJs, and $JJ_{64MP}$ is estimated to $23,713 + 260,000 = 283,713$ JJs by Eq. (4.3)).

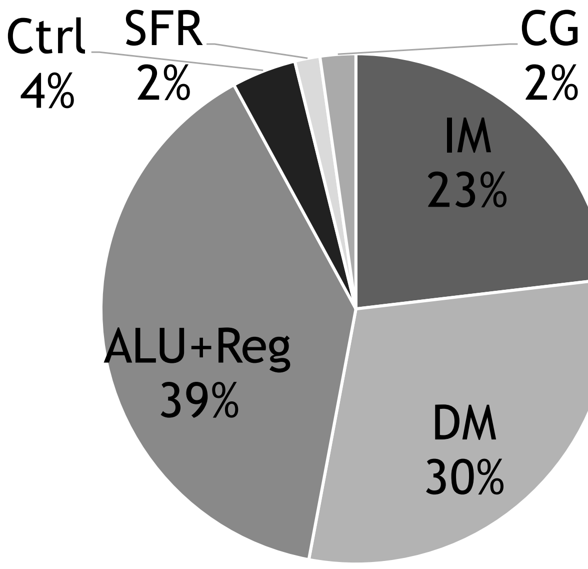


Figure 4.13: Power consumption breakdown of the processor

### 4.4.4 Estimating the energy efficiency of 64-bit processor

We compare the SFQ processor and CMOS processor models in terms of energy efficiency to evaluate the effectiveness of the SFQ processor. Based on the estimation results, we evaluate the energy efficiency of the 64-bit processor with the cooling cost. We use OPS per watt (OPS/W) for energy efficiency metric.

$$\text{OPS/W} = \text{OPS}/P \tag{4.6}$$

In this evaluation, we assume that 64-bit processor employs energy efficient SFQ circuit technology called ERSFQ [43] and 0.3 $\mu$m Nb process technology. We calculate them from the parameters of a 1.0$\mu$m Nb process [57] by applying the scaling rule for JJs. As a result, the clock frequency of the 64-bit processor with 0.3 $\mu$m Nb process is estimated to 107 GHz.

In ERSFQ logic, theoretically, although the dynamic power consumption becomes twice as conventional SFQ logic, the static power consumption is zero [54]. This is because ERSFQ provides the bias current using JJ with inductors (i.e., bias resistors are replaced to bias JJ); it does not consume static power, but the number of JJs increases (i.e., twice higher dynamic energy per switching). We assume that our processor employs ERSFQ without any configuration change. The

Table 4.4: Parameters

| Parameters | Values |
|---|---|
| $\Phi_0$ | 2.07 $(\mathrm{mV} \cdot \mathrm{ps})$ |
| $I_c$ | 0.1 (mA) |

Table 4.5: CMOS processor model's parameters

| | |
|---|---|
| Clock frequency | 5 GHz |
| The number of pipeline stages | 26 stages |
| The number of operations per cycle (64-bit integer) | 2 |
| Power consumption | 11 W |

power consumption of the 64-bit processor using ERSFQ, $P$ is given by Eq. (4.7),

$$P = 2 \times \alpha \Phi_0 I_c f \times JJ_{64MP} \tag{4.7}$$

where, $\alpha$ is switching probability, $\Phi_0$ is magnetic flux quantum, $I_c$ is the critical current of a JJ, and $f$ is clock frequency. Because $\alpha \Phi_0 I_c f \times JJ_{64MP}$ represents the dynamic power consumption, '2' is multiplied for ERSFQ power estimation. Some parameters are shown in Table 4.4.

By Eq. (4.7), the power consumption is estimated to 12.5 mW with the worst case of switching probability, i.e., $\alpha = 1$. Besides, we evaluate the power consumption of the whole system, including the cost of the cryocooler, which is a specific cooling system for SFQ circuits. The cooling cost is hard to model because the cooler's energy loss accounts for the majority of the whole cooler's energy. On the other hand, the coefficient of performance (COP), which is a ratio of efficient cooling provided to work required, indicates the efficiency of the cooler [34]. Therefore, we refer to the COP of two different cryocoolers: Gifford-McMahon (GM) cooler and Claude cooler [34].

For CMOS processor models, we refer to the parameters of the clock frequency, power consumption, and the number of pipeline stages (shown in Table 4.5) from Cell Broadband Engine Synergistic Processor Element (90nm) [14, 26]. After these processors, CMOS processors tend to improve multi-thread performance by increasing the number of processing cores and decreasing each clock frequency to

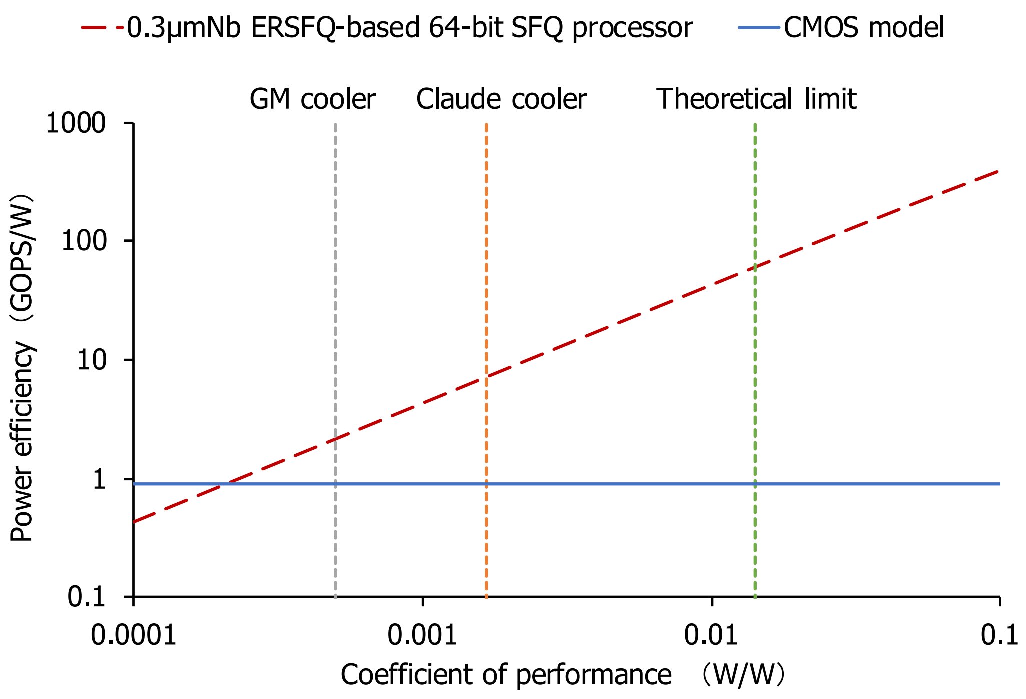


Figure 4.14: Power efficiency comparison between 64-bit SFQ processor and CMOS processor model

reduce the total power consumption. This evaluation uses the Cell processor's parameters because it is a representative processor following in-order execution and a deep pipeline structure (26 stages) to achieve high clock frequency. Note that this comparison is not the strict comparison between the SFQ processor and the Cell processor.

Fig. 4.14 shows the estimation results. The y-axis shows the energy efficiency, namely OPS/W, and the x-axis shows the COP. Fig. 4.14 displays the COP of two cryocoolers and theoretical limit as a guide. The energy efficiency of the 64-bit SFQ processor is estimated to 2.1 GOPS/W and 7.1 GOPS/W with GM cooler and Claude cooler, respectively. As a result shown in Fig. 4.14, the energy efficiency of the SFQ processor is about 2.3 and 7.8 times better than that of the CMOS model. The result shows that our SFQ processor has the potential to achieve high performance and power efficiency even with the cooling cost.

## 4.5 Conclusions

This chapter has designed and implemented a 4-bit SFQ processor as a prototype and confirmed the correct operation by real chip measurement to clarify our proposed architectural design guidelines' effectiveness. As a result, the prototype chip has successfully operated at 32 GHz with 6.5 mW. Based on these results, we have estimated the 64-bit processor and evaluated the power efficiency, including a cryocooler cost for 4 kelvin. The power efficiency with the cooling cost is estimated to at most 7.1 GOPS/W, and our processor outperforms the CMOS processor model 7.8 times. The result indicates that our SFQ processor based on proposed architectural design guidelines has the potential to achieve high performance and power efficiency even with the cooling cost. Although these results only show the high potential of the proposed architecture by evaluating peak performance, the effective performance of the SFQ processor is still unclear. Moreover, our processor adopts fine-grained multithreading to conceal almost all pipeline stalls for achieving high performance. Although the multithreading can keep the circuits simple (i.e., achieving higher clock frequency), the target applications are limited, and selecting the target application carefully for achieving high performance is necessary. Therefore, we must select the appropriate applications suitable for such SFQ characteristics for achieving high-performance SFQ computing.

This prototype processor consists of 23,713 JJs on $4.1 \times 5.3\text{mm}^2$ area, and this is one of the largest demonstrated circuits designed and implemented without any design automation tools. Although the circuit scale is limited to a few tens of thousands of JJs due to the chip area constraint, if the fabrication process technology evolves in the future, it is easily expected that it will be quite hard to design manually. Especially, the detailed timing adjustment for SFQ pulses is one of the critical issues for achieving large-scale high-performance SFQ circuits. Thus, placement and routing automation will significantly contribute to the growth of SFQ design. For the future development of SFQ computing, it is essential not only to study architecture but also to develop design automation tools, and we believe that our work highly motivates industry and academia to work on SFQ design automation technology.

# Chapter 5

# Extremely fast SFQ neural processing unit architecture

## 5.1 Introduction

We are now facing the era where both Moore's Law [65] and Dennard scaling [22] do not hold anymore. In this era, we are running out of a convincing option to improve the performance of the computer system, while maintaining its power and temperature budget. Therefore, we believe that it is the right time to actively exploit emerging device technologies with significant potentials and make a serious effort to improve their feasibility by resolving their limitations.

Among several candidates, superconductor SFQ logic family [49, 54] is a highly promising solution thanks to its ultra-fast speed and low-power consumption at 4 K. The SFQ technology enables a low-level voltage impulse-driven switching which allows both extremely-fast switching and low-energy consumption ($10^{-19}$ J per switching) [49, 54]. That is, with this technology, it is feasible to improve the device's clock frequency (and thus performance) by order of magnitude (i.e., several tens of GHz [55, 56]).

By focusing on these high potentials, many serious SFQ-related research efforts have been made in various aspects to promote the technology and automate its device and circuit-level design process (e.g., technology hardening, low-cost fabrication, design tool development) [59, 60]. As a result, the SFQ logic is now considered for an extreme-performance computing and a promising post-Moore solution.

However, due to its unique pulse-driven nature, SFQ logic requires completely different architecture designs from conventional CMOS technology. Therefore, with the architectural trade-offs considered, the following questions must be clearly addressed to computer architects: (1) what architecture is promising for this technology, (2) how to implement various microarchitectural units with the voltage pulse-driven logics, (3) how to maximize its potential at the architecture level while minimizing its limitations, and (4) how to simulate and validate a proposed architecture design.

In this paper, we resolve the fundamental challenges by (1) providing straightforward answers to the questions above, and (2) presenting *SuperNPU*, our example SFQ-based neural processing unit (NPU) design. First, as our case-study architecture in this work, we choose to architect a conventional NPU and present the basic structure with carefully designed microarchitectural units. For instance, we design our baseline NPU architecture consists of processing elements (PEs) with the weight-stationary dataflow, systolic array network, and data alignment unit. This baseline NPU architecture well satisfies the requirements of SFQ-based logics such as fast computation, dataflow-like data movements, and shift-register-based memory implementation, respectively.

Next, we implement an architecture-level simulation framework to model an SFQ-based NPU architecture accurately. Our simulator can accurately estimate the under-the-design NPU's performance, power consumption, area at various levels (i.e., gate, microarchitecture, architecture). For the purpose, the simulator constructs a target SFQ-based NPU architecture by integrating SFQ-based microarchitecture and gate modules using AIST 1.0 $\mu$m fabrication process technology [57]. We carefully validate the simulator by comparing the results obtained from our die-level microarchitecture prototypes and post-layout simulations against our modeling results.

Third, based on the validated model, we identify key performance bottlenecks in a naively designed SFQ-based NPU. First, the data movement among different units and within a single unit takes too long, mainly due to the shifting register-based operation. Next, fast computing units often become idle due to the workload's low computational intensity and relatively slow memory access. Also, the on-chip memory underutilization can make the above overheads much worse.

Lastly, we present *SuperNPU*, our example SFQ-based NPU design, which effectively resolves the performance bottlenecks at the architecture level. First, it merges the partial-sum and output memories to avoid unnecessary inter-memory data movements. Second, it partitions a larger on-chip buffer to multiple small chunks to reduce the length of intra-memory shifting as well as the underutilization. Third, it increases the computational intensity by balancing hardware resources for a larger-batch purpose. Fourth, it further increases each PE's utilization by assigning more registers to each PE for enabling multi-kernel execution.

Our evaluation shows that SuperNPU significantly outperforms a conventional NPU design by 23 times when running various CNN workloads. However, without the SFQ-aware architectural optimizations, the SFQ-based NPU design's performance drastically drops to the point even below the conventional design. Therefore, it is extremely essential to identify the SFQ-unfriendly bottlenecks and architect an optimized design to resolve them. With the cooling cost considered, SuperNPU's performance per watt is slightly higher than the conventional design. But, with free cooling cost assumed, SuperNPU's performance per watt becomes significantly higher than the conventional design by 490 times.

In summary, our work makes the following contributions:

- **Architecting an SFQ-based NPU:** To the best of our knowledge, this is the first work to design an NPU which addresses the SFQ technology's architectural trade-offs.
- **Simulation framework:** It is also the first work to model and validate a simulator for SFQ-based architectures.
- **SFQ-specific architectural optimizations:** We identify critical architectural bottlenecks and optimizations which can cause a performance variance around 60 times.
- **Significant results:** SuperNPU provides extreme performance and power efficiency by outperforming a conventional design by 23 times and 490 times,

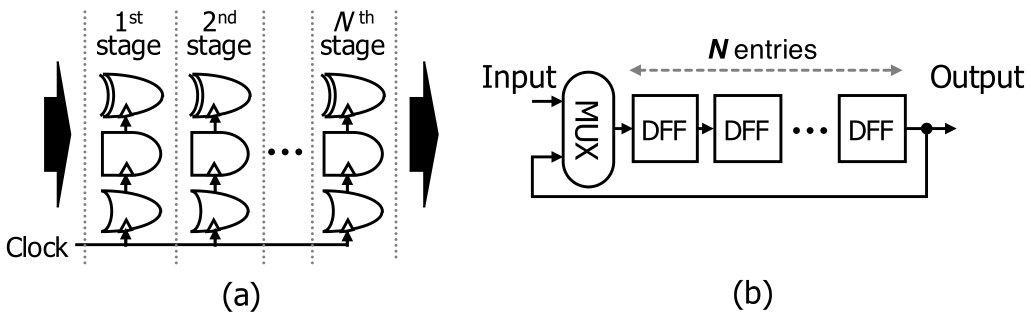


Figure 5.1: Example of the SFQ technology's architectural characteristics (a) Gate-level-pipelined datapath (b) 1-bit $N$-entry shift register

respectively.

- **Applicability:** Our modeling-driven methodology can be applied to other architectures favoring the SFQ logic.

## 5.2 Background & Motivation

### 5.2.1 SFQ technology in the architect's perspective

In the architect's perspective, it is essential to understand SFQ logic's architectural characteristics originated from the pulse-driven nature. Therefore, we summarize some notable features as follows.

#### Deeply pipelined datapath

In the SFQ logic, architects can naturally apply the gate-level pipelining without any overhead. All SFQ logic gates are synchronized with the clock because they need a clock pulse to transfer the stored SFQ to the adjacent gates. In other words, every SFQ gate has the latch functionality and thus can be pipelined without additional DFFs (Fig. 5.1(a)). With this property, several chips have been successfully demonstrated at several tens of GHz [55, 56]. However, the deep pipeline structure can suffer from performance degradation because it is difficult to avoid data (or control) hazards and the huge pipeline stalls. Therefore, the SFQ technology favors streaming execution rather than applications with complex control flows.

### Frequency determination with pulse-driven clocking

Unlike conventional CMOS technology, SFQ circuits' frequency is determined by the timing difference between the data and clock pulse arrival. In the CMOS technology, the clock frequency is bounded by the longest datapath delay because it only can put single digital information (i.e., voltage level) in a wire. On the other hand, SFQ logic can put several data into a single wire because its data is encoded as a voltage pulse. That is, SFQ circuits can achieve high frequency by flowing many data pulses through a single wire, simultaneously. However, if there is a large difference between the data and clock arrival timing for an SFQ gate, its frequency can be significantly reduced. This is because the next clock pulse should wait for the slow data pulse propagation, and thus the time interval between two adjacent clock pulses increases. Therefore, it is crucial to match the data and clock pulse arrival timing for maximizing the SFQ circuits' frequency.

### Shift-register-based on-chip memory

For the SFQ logic's on-chip memory, the shift-register-based memory is much more practical than the random access memory (RAM). Even though we can implement RAM with SFQ technology, it severely suffers from low driving capability and scalability. Such limitations mainly result from the difficulty of driving the word lines and bit lines with the small pulses [49, 88]. On the other hand, a shift-register-based memory does not have those problems because it just consists of the serially connected DFFs and the feedback loop (Fig. 5.1(b)). However, it is difficult for the shift-register-based memory to support the random memory access due to the complex control logic and the variable access latency [37]. Therefore, the SFQ technology favors applications with sequential memory access when its on-chip memory implementation is considered.

### Lack of off-chip memory technology

It has been a long-standing challenge to implement a large-scale and high-speed off-chip memory operating at the 4K environment. There has been a few research about JJ-based memories [73, 45, 76], and one of them is Vortex Transition Memory (VTM) [73]. The VTM is the largest Josephson memory whose 4-kbit prototype has been demonstrated. Despite the demonstration, it has been difficult to practically use the VTM mainly due to the scaling and speed problems with the AC-biasing and the large superconductor-ring-based memory cells. Even though several off-chip memory technologies (e.g., hybrid Josephson-CMOS memory [45, 76], Josephson magnetic memory [21]) are currently being developed, these technologies also have not been put to practical use yet. For these reasons, it is currently practical to use CMOS memory technology, which is slower than the 4 K JJ-based memory but large and reliable. Therefore, SFQ technology favors computation-oriented applications with a minimal number of off-chip memory access.

### 5.2.2 Challenges for designing SFQ-based architectural unit

Even though there have been several studies regarding the SFQ architecture's features [81, 23, 87, 77, 55], there still exist critical challenges for designing SFQ-based architectural units.

**SFQ-optimal architecture design**: First, for the target architecture application, architects should carefully design each microarchitectural unit because the novel circuit-level trade-offs occur in SFQ logic (e.g., frequency trade-off with the applied clocking scheme). Furthermore, architects must carefully analyze the performance bottlenecks and propose the best SFQ architecture design based on the analyses. However, as far as we know, there does not exist either such SFQ-friendly microarchitecture implementation or the SFQ-optimal architecture proposed with the bottleneck analysis.

**Absence of an SFQ-based architecture modeling tool**: Architects are in dire need of high-level architecture modeling tools to design and evaluate their architectural innovations, especially for emerging technologies such as SFQ logic devices. Even though researchers recently have made an effort to develop several SFQ design automation tools [59, 60], to the best of our knowledge, a reliable SFQ-based architecture modeling tool is currently absent.

### 5.2.3 Research goal: Provide SFQ design principles with NPU

In this paper, we resolve the challenges and provide the guidelines for designing an SFQ-optimal architectural unit by presenting an extreme-performance SFQ-based NPU. We first conduct thorough analyses and introduce the baseline SFQ-based NPU architecture by designing all microarchitectural units in the SFQ-friendly manner (Section 5.3). Next, on top of the baseline NPU architecture, we develop *SFQ-NPU*, a validated SFQ-based architecture modeling tool (Section 5.4). Finally, we use the tool to identify critical performance bottlenecks and propose our SFQ-optimal NPU, which successfully resolves the bottlenecks at the architecture level (Section 5.5).

In this work, we choose NPU as one of the promising examples to apply our design principles for the following reasons. First, there are no complex control flows in Deep Neural Network (DNN) applications, and therefore we can fully exploit the SFQ's gate-level pipelining nature without control hazard. Second, we can take the best advantage of shift-register-based memory and avoid its disadvantage thanks to the static memory access pattern of DNN algorithms. Finally, NPUs can reduce off-chip memory access by utilizing the data-reuse pattern in DNN applications. Note that the underlying SFQ circuits, such as multipliers and adders, have already been demonstrated with around 50 GHz frequency [55, 56]. Besides, we currently target the DNN inference as the first case study to show SFQ-based NPU's potential.

## 5.3 Baseline SFQ-based NPU design

In this section, we design the baseline SFQ-based NPU architecture by identifying the SFQ-friendly implementation for key microarchitectural units. Fig. 5.2 shows the overview of our baseline SFQ-based NPU which mainly consists of four microarchitectural units: on-chip network unit (NW unit), processing element (PE), data alignment unit (DAU), and on-chip buffers. We perform detailed circuit-level analyses to describe our design choice for each unit, except for the shift-register-based on-chip buffers explained in Section 5.2.1.

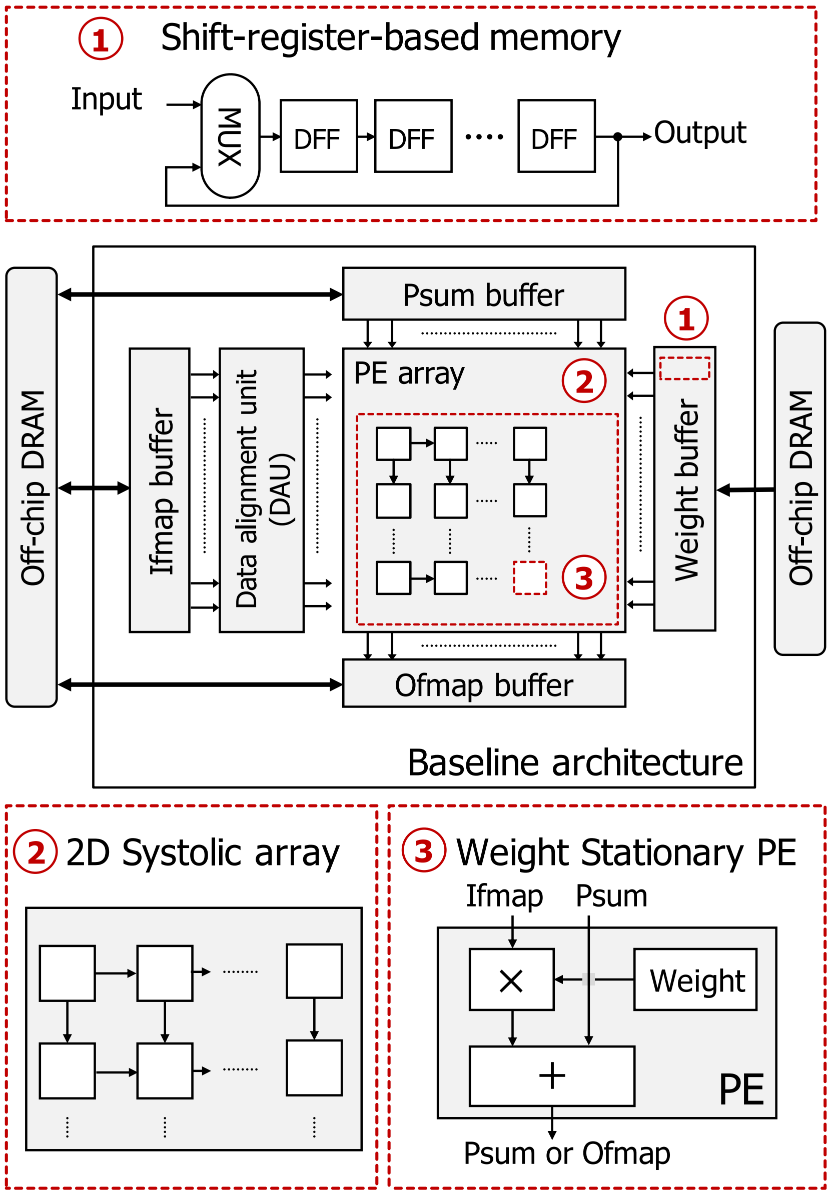


Figure 5.2: Overview of our baseline SFQ-based NPU design

### 5.3.1 On-chip network unit design

To design the SFQ-friendly on-chip network, we compare two representative network unit (NW unit) designs: fan-out network and store-and-forward chain. The fan-out network multicasts the data to several PEs simultaneously by using the bus or tree structure. On the other hand, the store-and-forward chain provides the data and subsequently forwards it from a PE to the adjacent PE. Note that a network branch consists of a DFF (D in Fig. 5.3) and a wire component called splitter (S in Fig. 5.3), which splits a pulse into two identical pulses.

Among these two network designs, we adopt the store-and-forward chain because it is superior to the fan-out network in terms of both clock frequency and area. Fig. 5.3 shows the structures of three network design candidates: two splitter tree (2D and 1D) designs (fan-out network) and a 2D systolic array (store-and-forward chain). In our analysis, we include both 2D and 1D splitter tree designs which can be applied to the output stationary (OS) and weight stationary (WS) dataflow, respectively. Also, we assume that all network designs target 2D square-shaped PE array.

Fig. 5.4 shows the critical-path delay (i.e., the inverse of maximum frequency) and the area comparison for the three network designs, obtained with JSIM [25]. First, the 2D splitter tree significantly suffers from the long critical-path delay due to the increasing timing difference of two PE inputs. As shown in Fig. 5.3(a), a single PE requires two inputs from each splitter tree. As both splitter trees share a global clock line, the critical-path delay increases in proportion to the PE array width (Input arrival timing in Fig. 5.3(a)). As a result, the critical-path delay of the 2D splitter tree keeps increasing with the PE width and reaches above 800 ps in $64 \times 64$ PE array. Even though we can mitigate this problem with the aggresive clock skewing (i.e., intentionally increase the clock propagation delay in path ❶), it incurs much more area overhead and lowers the yield of fabrication [75]. Next, even if there is no such a timing issue in the 1D splitter tree, its area overhead is high as the same with the 2D tree. The large area overhead is mainly due to the large number of wire cells for the tree construction.

On the other hand, the 2D systolic network has the shortest critical-path delay and the smallest area, as shown in Fig. 5.4. Even though the 2D systolic network

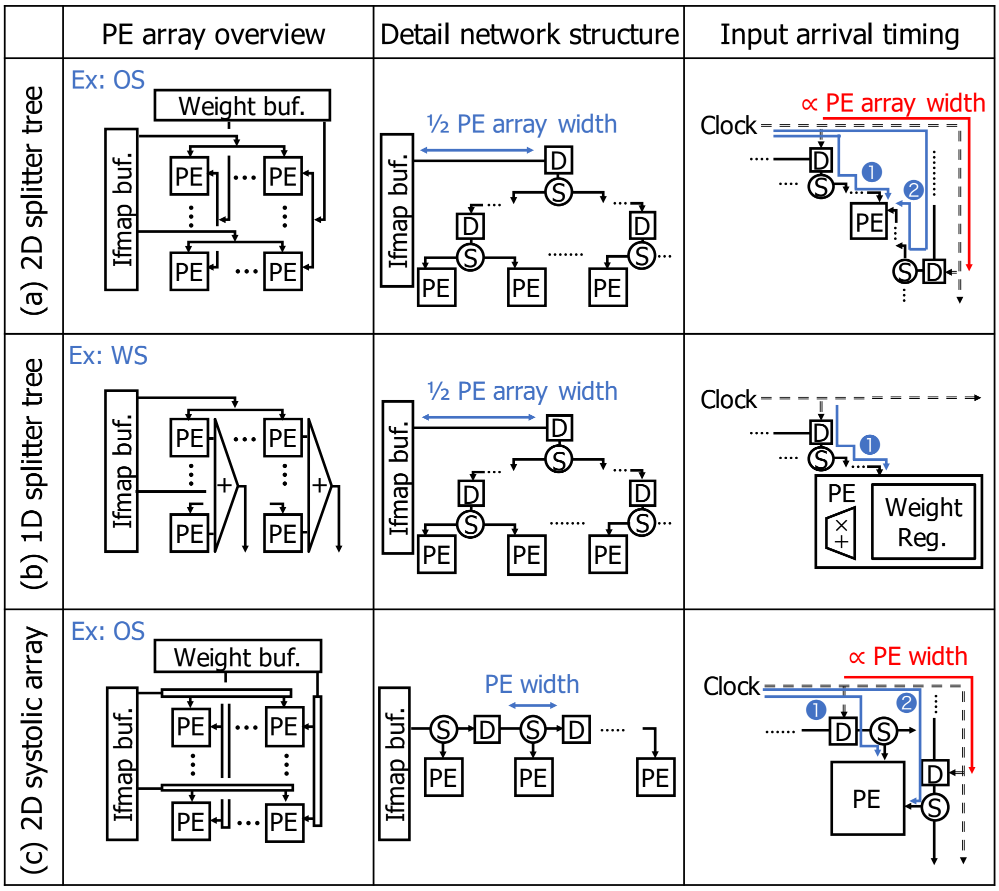


Figure 5.3: On-chip network structure for three alternative designs

also provides two different inputs to a single PE as same with the 2D splitter tree, their timing difference is negligible (Fig. 5.3(c)). Besides, its simple structure does not require much wire cells. For these reasons, we conclude that the systolic array is more suitable and adopt it as our on-chip network design.

### 5.3.2 PE design

For the SFQ-friendly PE design, we identify the most suitable dataflow by carefully considering the SFQ logic's circuit-level characteristics. Among three major dataflows in a 2D systolic network, *Weight Stationary* (WS), *Output Stationary* (OS), and *Input Stationary* (IS) [16, 63], we focus on WS and OS because the PE with IS has almost the same hardware structure as the PE with WS. Fig. 5.5(a) shows the PE with WS dataflow, where PE holds a weight in its register, multiplies

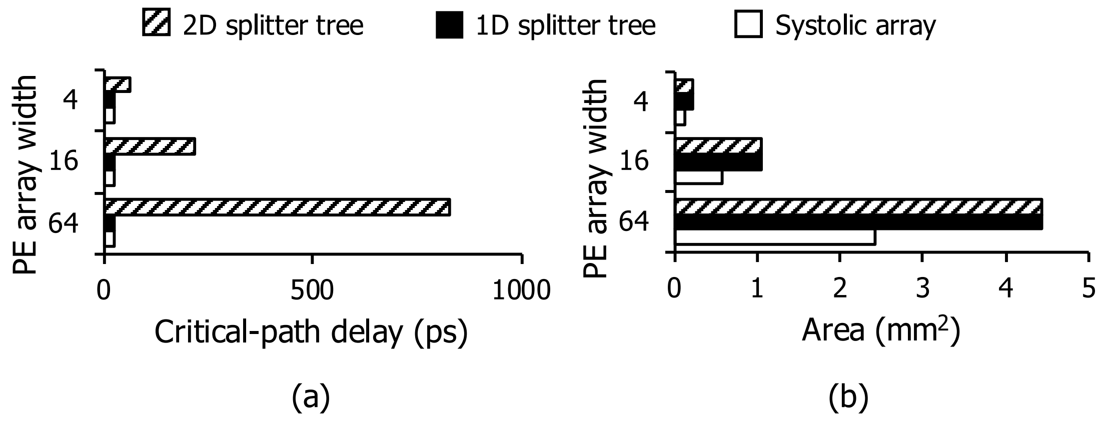


Figure 5.4: Network unit designs' (a) critical-path delay and (b) area comparison

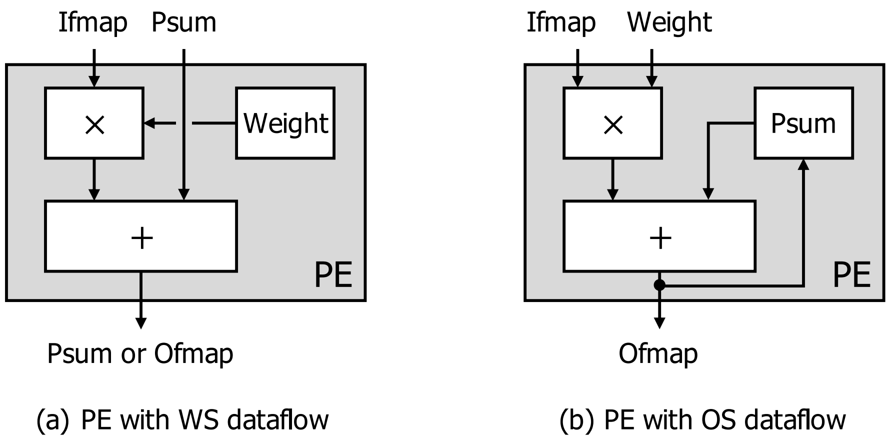
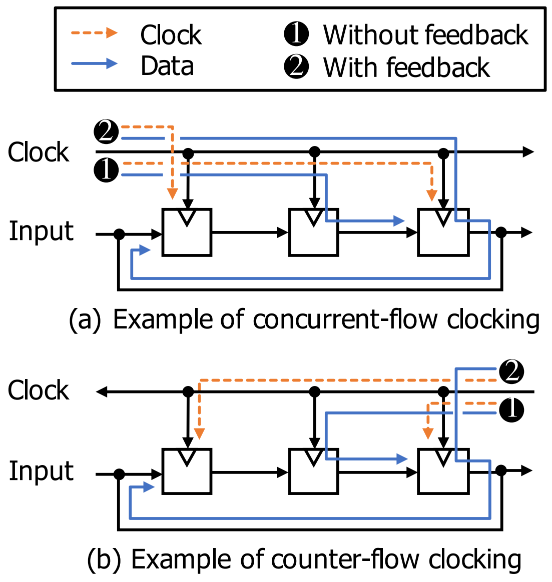


Figure 5.5: PE designs with two different dataflows

it with the input feature map data (ifmap), and adds the result to the partial sum input (psum). On the other hand, the PE with OS dataflow has a feedback loop consisting of the adder and its register, and continuously accumulates the partial sums to generate final output feature map data (ofmap) (Fig. 5.5(b)).

Among these two PE designs, we choose the PE with WS to maximize the clock frequency because it does not include any feedback loop. Unlike the CMOS technology, the existence of the loop significantly degrades the SFQ circuit's frequency as the loop enforest the slower clocking scheme.

Fig. 5.6 provides the example with two representative clocking schemes: (a) concurrent-flow clocking and (b) counter-flow clocking. In our example, we show


Clock
Data
❶ Without feedback
❷ With feedback
Clock
Input
❶
❷
(a) Example of concurrent-flow clocking
Clock
Input
❶
❷
(b) Example of counter-flow clocking

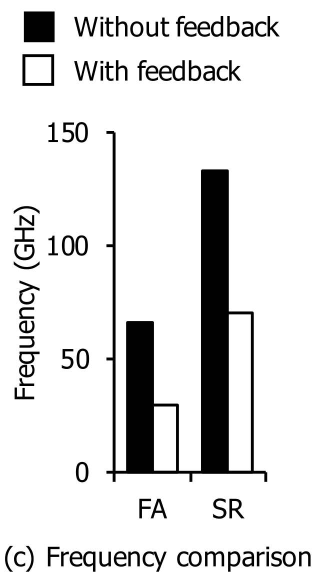


Figure 5.6: Feedback loop's impact on the frequency of SFQ circuits

how the SFQ circuit's frequency is affected by the feedback loop. As Fig. 5.6(a)❶ shows, when there is no feedback loop, SFQ circuits can hide the data propagation delay by flowing the clock pulse along with the data. However, such clocking cannot be utilized when the circuit includes the feedback loop. In fact, the circuit's frequency is significantly reduced because the next clock pulse should wait for a very long data transfer through the feedback path (Fig. 5.6(a)❷). On the other hand, we can resolve this problem with the counter-flow clocking, which can perfectly hide the data feedback delay (Fig. 5.6(b)❷). However, the frequency of the counter-flow clocked circuit is much lower than that of the concurrent-clocked circuit without feedback. Such difference is due to the unhidden feed-forward delay of the counter-flow clocking (Fig. 5.6(b)❶).

Fig. 5.6(c) shows the feedback loop's impact on the SFQ circuit's clock frequency by running JSIM [25] simulations with simple example circuits, a full adder (FA), and a shift register (SR). For the circuits without the feedback loop and with the loop, we apply the concurrent-flow clocking and counter-flow clocking, respectively. As Fig. 5.6(c) clearly shows, the existence of the feedback loop significantly degrades the clock frequency, from 66 GHz to 30 GHz in FA and from 133 GHz to 71 GHz in SR. Thus, we conclude that the PE design without a feedback loop, PE with WS, is a more SFQ-friendly choice and adopt it as our PE design.

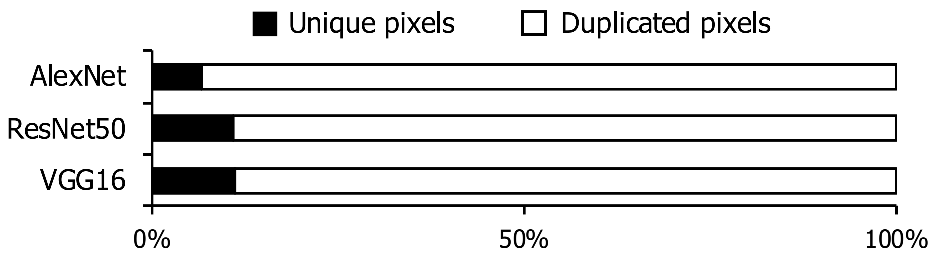


Figure 5.7: Data ratio breakdown for unique and duplicated ifmap pixels

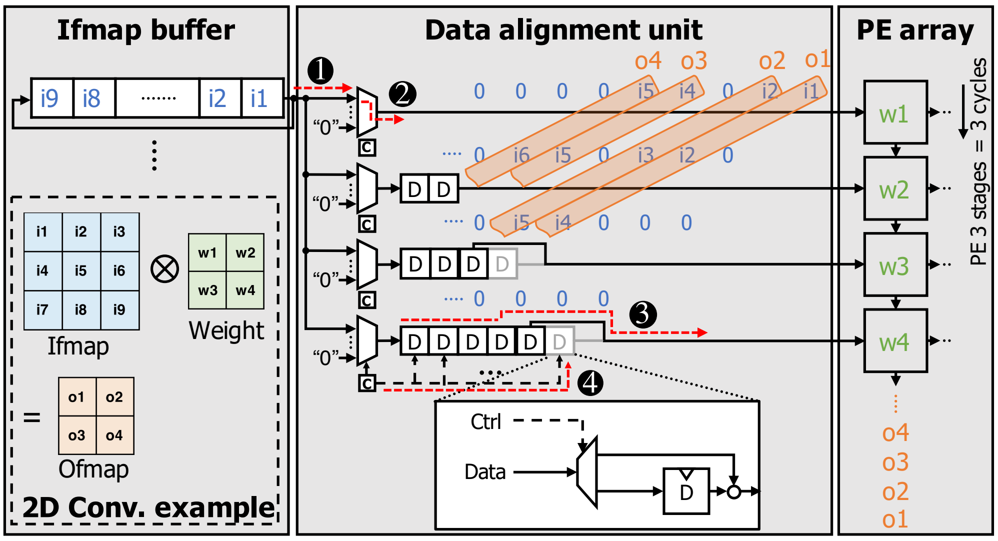


Figure 5.8: Data alignment unit's structure with the working example

### 5.3.3 Data alignment unit design

In the SFQ-based NPU adopting systolic network and WS dataflow, the ifmap buffer can suffer from a large amount of duplicated data. As the ifmap buffer is the shift-register-based memory, each ifmap buffer row dedicatedly feeds data to the corresponding PE array row. However, in CNN execution, weights mapped to the adjacent PE array rows require partly the same ifmap data due to the weight sharing property of CNN. Therefore, the duplicated data significantly wastes the buffer capacity if adjacent ifamp buffer rows hold all ifmap data shared across the different weights. Fig. 5.7 clearly shows that the amount of duplicated data can be over 90% for three CNN networks. Note that such a massive waste of on-chip buffer capacity incurs a severe off-chip memory pressure.

To resolve the problem, we design a data alignment unit (DAU), which replicates and forwards data to the appropriate PE rows at exact timing. Fig. 5.8 shows the DAU's structure with a working example that runs a simple 2D-convolutional operation. Our DAU consists of sets of a selector, a controller, and cascaded special DFFs for each PE row. The DAU operates in two steps: 1) data selection and 2) timing adjustment.

**Data selection**: Before starting computation, each ifmap buffer row dedicatedly holds data for a given ifmap channel. First, each ifmap buffer row provides its data to all DAU rows through a splitter tree, where each DAU row is dedicated to a single PE row's weight. For example, nine ifmap pixels are transferred to all four DAU rows in Fig. 5.8 (❶). Next, the selector in each DAU row selectively takes the required input for the weight mapped in the corresponding PE array row. The first row in Fig. 5.8 takes only i1, i2, i4, and i5, and 0 for others as a bubble to avoid the computation stall (❷). The bubbles are filtered at the end of computation by using a valid bit. For such data selection, the controller in each DAU row dynamically generates control signals. Note that the controllers can identify whether the given input is required or not based on the DNN layer configuration and current weight mapping information (e.g., current ifmap and weight pixel index).

**Timing adjustment**: To adjust the arrival timing of selected ifmap pixels, DAU utilizes the cascaded special DFFs with different lengths. By using the DFFs, each DAU row delays to feed the data because the computed psum in the above PE and the ifmap data should simultaneously arrive at the PE. For example, if the PE consists of three pipeline stages, the inputs from the second row should be delayed at most 2 ($= 3 - 1$) cycles. In fact, our 8-bit PE consists of 15 pipeline stages. Also, we bypass some DFFs when the adjacent PE rows map the weights with different row index. For example, as the weight's row index mapped to the third PE increases from that of the second PE (from 1 (w2) to 2 (w3)), we should bypass one DFF for the correct operation (❸). To support the bypassing, our special DFF has a bypassing line, whose control signal is statically determined by weight filter width, strides, and current weight index (❹).

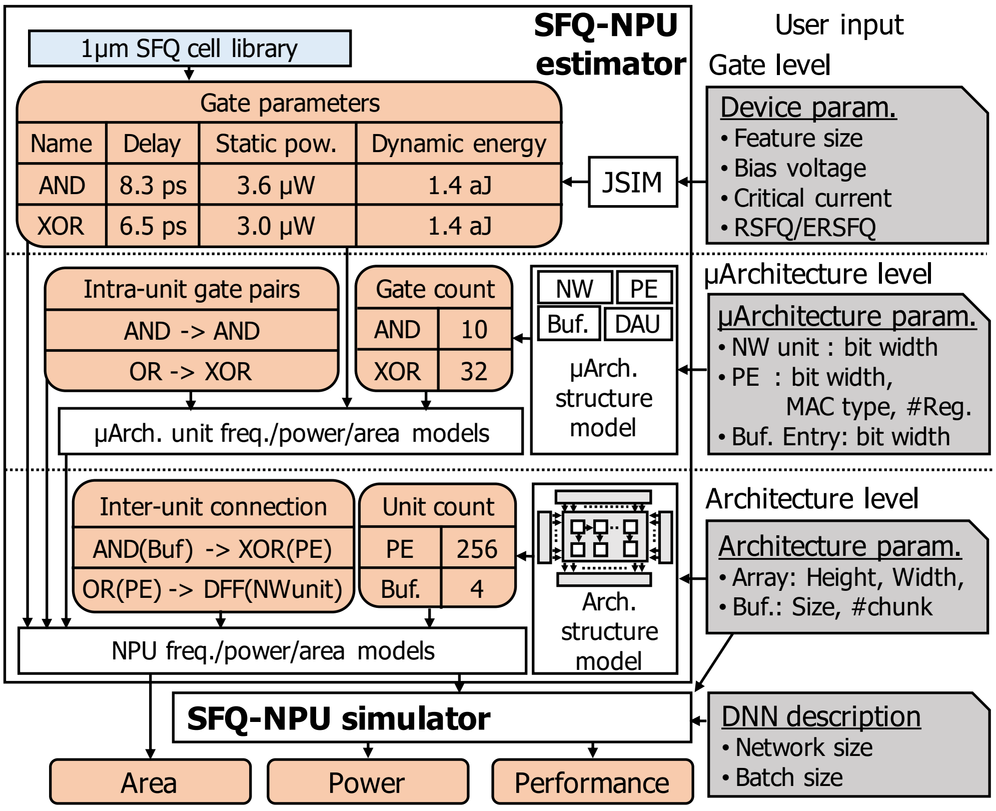


Figure 5.9: SFQ-NPU overview

## 5.4 Simulation framework

In this section, we describe our architectural simulation framework, *SFQ-NPU*, to explore and optimize the SFQ-based NPU architecture. Fig. 5.9 shows the overview of SFQ-NPU, which consists of two simulation engines: *SFQ-NPU estimator* and *SFQ-NPU simulator*. SFQ-NPU estimator takes device-level, microarchitecture-level, architecture-level information as inputs, and derives the frequency, power, and area of the target NPU design. Based on the obtained frequency and power information, the SFQ-NPU simulator reports the effective performance and power consumption by simulating target DNN applications. In the following sections, we explain the implementation details of each engine.

### 5.4.1 SFQ-NPU estimator

To carefully consider the SFQ logic's unique features ranging from the device to architecture, our SFQ-NPU estimator takes a strategy of three-layer abstraction:

gate-level, microarchitecture-level, and architecture-level estimation.

### Gate-level estimation

The gate-level estimation layer accurately provides the timing parameters (i.e., *SetupTime*, *HoldTime*, and delay), the power information (i.e., static power and access energy), and the area for all SFQ logic gates and wire cells with the given device parameters (e.g., bias voltage, critical current). The gate models are compatible with two SFQ technologies; rapid single-flux-quantum (RSFQ) [49, 86, 89, 78], and energy-efficient RSFQ (ERSFQ) [43]. RSFQ is the most practical and proven technology in the successful demonstrations, whereas ERSFQ is a promising technology that completely excludes the static power dissipation. The only difference is how to supply the DC bias current, i.e., RSFQ uses the bias resistors; on the other hand, ERSFQ uses the bias JJs.

For the RSFQ gates, we extract all gate parameters by running JSIM [25] simulations with RSFQ cell library for AIST 1.0$\mu$m fabrication process technology [57]. For example, the access energy is derived by taking an average of the dynamic energy for all the possible states. Besides, we calculate each gate's area based on its number of JJs.

On the other hand, we estimate gate parameters of ERSFQ based on those of RSFQ gates due to the lack of fabrication information (or cell library) about ERSFQ technology. Specifically, the timing parameters and area of ERSFQ gates are assumed to be the same as those of RSFQ because all their gate structures are the same, except for the bias current supply line. Meanwhile, we estimate the access energy and static power of ERSFQ gates as twice as that of RSFQ and zero, respectively. Note that the difference in both access energy and static power originates from the JJ-based DC biasing scheme [43].

### Microarchitecture-level estimation

This abstraction layer estimates the frequency, static power, access energy, and area of each microarchitectural unit designed in Section 5.3 (i.e., NW units, PE, DAU, and on-chip buffers). For the accurate estimation, the microarchitecture-level layer first generates the intra-unit gate pair and the gate count information

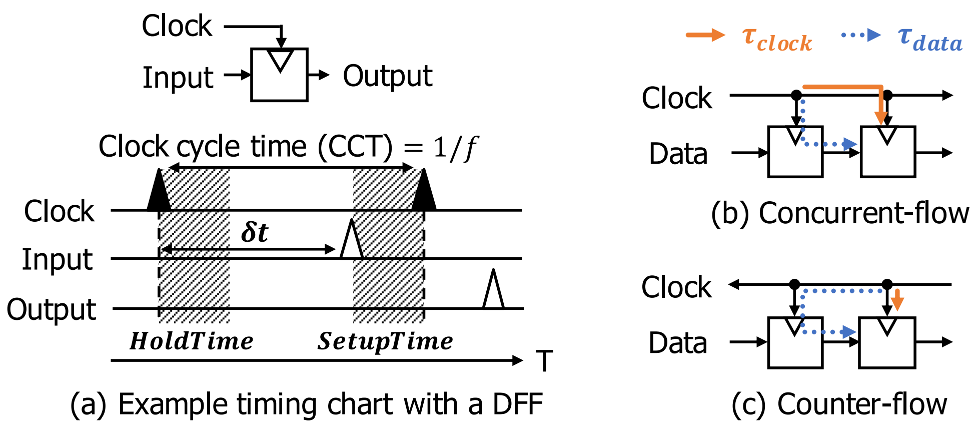


Figure 5.10: Frequency model illustration

for each unit based on the gate-level circuit structure model. The intra-unit pair and the gate count are utilized to derive each unit's frequency and power/area, respectively.

$$f = 1/\mathrm{CCT} = 1/(\mathit{SetupTime} + \mathrm{Max}(\mathit{HoldTime}, \delta t)) \tag{5.1}$$

With the gate-level pipelining nature considered, the microarchitecture-level frequency model calculates the frequency of all gate pairs in the target unit and takes the minimum value as the unit's frequency. Fig. 5.10 and Eq. (5.1) illustrate the model to calculate the frequency of one gate pair, where $\delta t$ is the difference between the data and clock propagation delay (i.e., $\tau_{data} - \tau_{clock}$). Note that Eq. (5.1) is the direct translation of the two timing constraints: 1) data should arrive after the HoldTime and 2) the next clock pulse should arrive after SetupTime elapsed from the data arrival.

To reflect the real-world SFQ circuit design practice, we model both representative clocking schemes, concurrent-flow clocking (Fig. 5.10(b)) and counter-flow clocking (Fig. 5.10(c)). As explained in Section 5.3.2, to achieve high frequency, we apply the concurrent-flow clocking to all circuits without the feedback loop. Also, we include the frequency-enhancing technique called clock skewing, which minimizes $\delta t$ by adjusting the length of data and clock line. On the other hand, we apply the counter-flow clocking to the circuits with the feedback loop, such as shift-register-based on-chip buffers.

Meanwhile, the microarchitecture-level power and area models calculate the

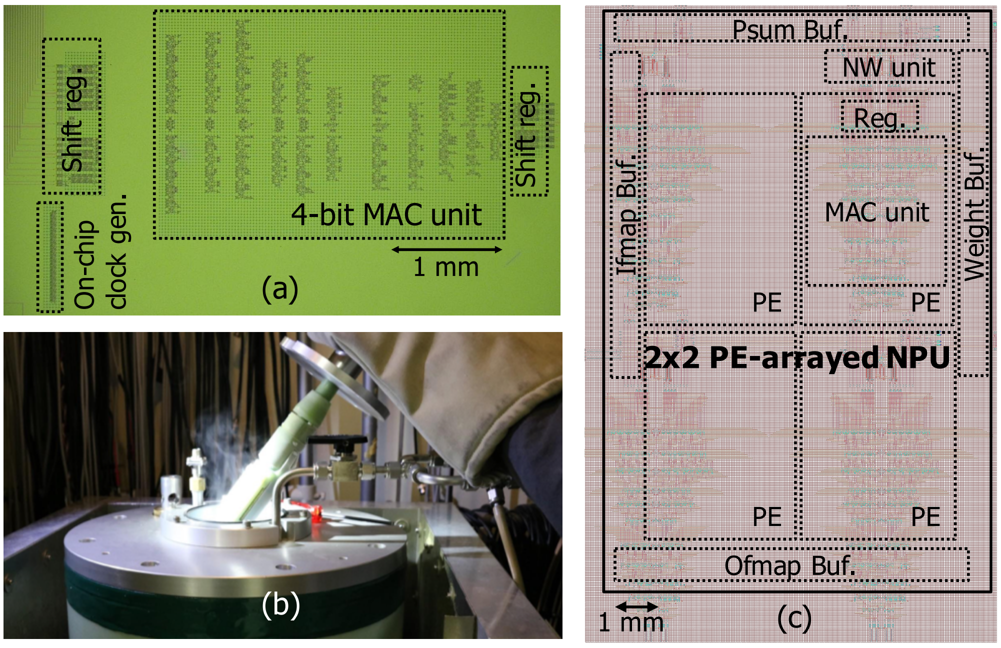


Figure 5.11: Model validation setup (a) Chip microphotograph of 4-bit MAC unit (b) 4 K measurement setup (c) Layout of the $2 \times 2$ PE-arrayed NPU

power information (i.e., static power and access energy) and area of each unit based on the gate count information and the gate parameters.

### Architecture-level estimation

The architecture-level layer reports the final estimation results regarding the area, static power, access energy, and clock frequency of the target NPU configuration. For the accurate prediction, this layer not only integrates the microarchitecture-level estimations based on the unit counts but also considers the inter-unit gate pair information. For instance, we calculate all the inter-unit communication latency based on the interfacing gates' timing parameters and include them to derive the highest frequency in NPU. Also, based on the estimated unit-to-unit distance, we calculate the area of wire cells required to connect each unit and include it to the final area estimation.

### Model validation

We carefully validate our SFQ-NPU estimator in terms of the frequency, power, and area by comparing it with a fabricated 4-bit MAC unit (Fig. 5.11(a)) measured

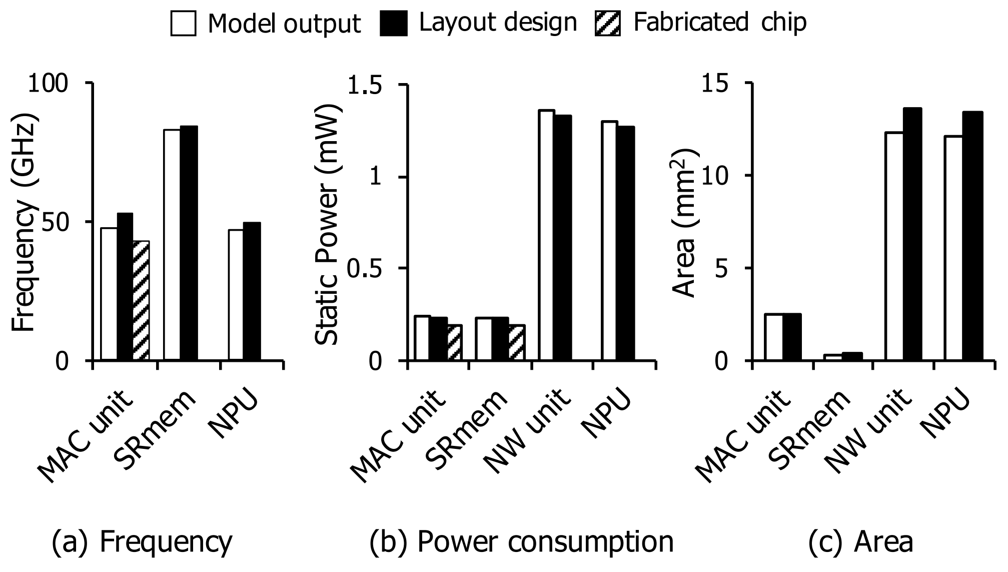


Figure 5.12: Model validation result

in the 4 K environment (Fig. 5.11(b)). Also, we compare the model's output with the post-layout characterizations for 8-bit 8-entry shift-register-based memory (SRmem), 8-bit NW unit, and a 4-bit $2 \times 2$ PE-arrayed NPU (Fig. 5.11(c)). Note that our gate-level estimation is already validated in its accuracy because it is based on the validated cell library, which succeeded in fabricating real chips for many times.

First, we validate our microarchitecture-level estimation with MAC unit, SRmem, and NW unit. Note that there is no frequency result for a single NW unit because it only consists of DFF-splitter pairs. As Fig. 5.12 shows, SFQ-NPU estimator accurately predicts all the frequency, power, area for each unit with the average error of 5.6%, 1.2%, and 1.3%, respectively.

Next, we also validate the architecture-level estimation with the 4-bit $2 \times 2$ PE-arrayed NPU design (Fig. 5.11(c)). Even though it is a small NPU prototype, the layout design is enough to show the inter-unit connections' impact on the frequency due to the 2D-systolic network's scalable structure. As shown in Fig. 5.12's NPU, our model well matches the frequency, power, and area result of the post-layout simulation with the error of 4.7%, 2.3%, and 9.5%, respectively.

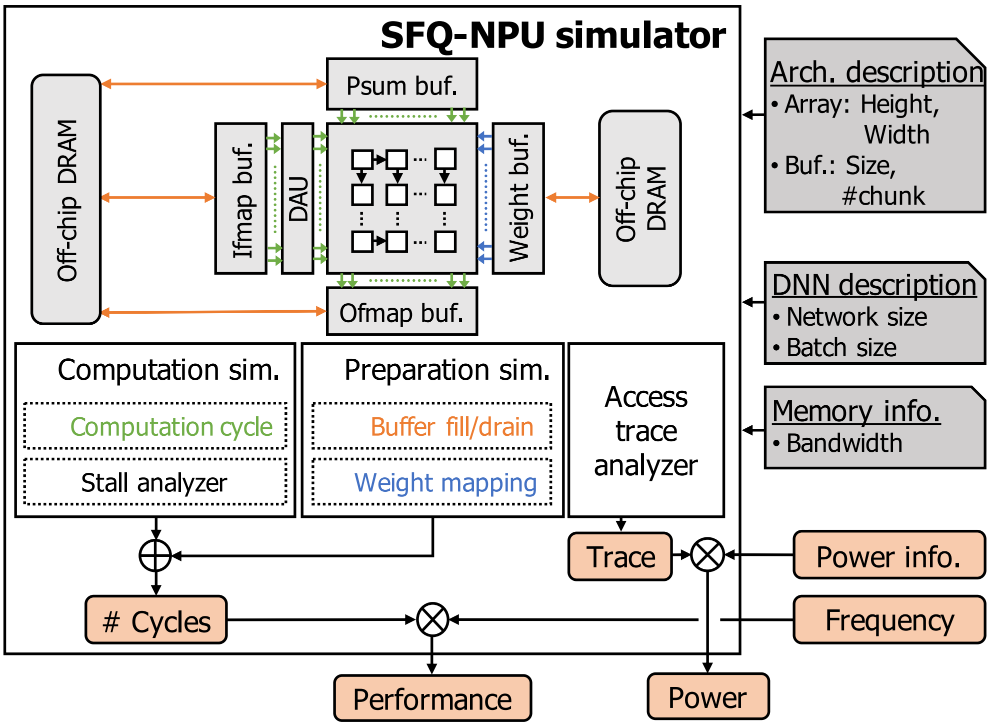


Figure 5.13: SFQ-NPU simulator overview

### 5.4.2 SFQ-NPU simulator

For the given SFQ-based NPU design running DNN applications, SFQ-NPU simulator reports the effective performance and power consumption based on the obtained frequency and power information from the SFQ-NPU estimator. As the first step for the simulation, SFQ-NPU simulator analyzes all required weight mappings by taking the DNN description file (i.e., ifmap window size, filter window size, the number of filters, the number of strides) and architecture description file as inputs. We use a batch of typical DNN input images ($224 \times 224 \times 3$) as inputs. Next, for each weight mapping, the simulator runs the cycle-based simulation to calculate the consumed cycles and the activated cycles for each hardware unit. During the simulation, the simulator also models the memory stall incurred by limited memory bandwidth by taking memory bandwidth as its input. Finally, the SFQ-NPU simulator aggregates the result of each mapping and reports the performance numbers (e.g., latency, throughput, PE utilization) and the power values (i.e., static power, dynamic power).

## 5.5 Optimizing SFQ-based NPU design

In this section, with our simulation framework, we architect an extreme-performance SFQ-based NPU by taking the best advantage of SFQ technology. Similar to other devices, SFQ-based NPUs have a large design space that cannot be easily explored even with the modeling tool. Therefore, we start from our baseline SFQ-based NPU architecture (Section 5.3) and identify the major performance bottlenecks to be resolved (Section 5.5.1). Next, we propose an optimal SFQ-based NPU architecture, *SuperNPU*, which resolves the identified bottlenecks with the architecture-level solutions (Section 5.5.2).

In the following sections, we conduct performance analyses by running six CNN workloads (i.e., AlexNet [46], FasterR-CNN [62], GoogLeNet [72], MobileNet [35], ResNet 50 [32], VGG16 [70]) with our simulation framework. As the input fabrication process information, we take the currently available AIST 1.0 $\mu$m process to show the SFQ technology's performance potential conservatively[1]. Moreover, we assume the memory bandwidth of 300 GB/s, which is the typical value of HBM used by the recent TPUv2 [2].

### 5.5.1 Design implications for the SFQ-optimal NPU architecture

#### Baseline SFQ-based NPU setup

We first introduce performance-side design implications by conducting analyses with the baseline SFQ-based NPU design introduced in Section 5.3 (hereinafter called *Baseline*). To show the implications, we start from Baseline following the TPU core's [39] architectural specification for three reasons. First, we target the server-side NPU due to the need for cryogenic cooling support. Second, Baseline has a similar hardware structure with the TPU core (i.e., weight-stationary dataflow and systolic-array network). Third, its estimated area might be comparable to the TPU core ($< 330$ mm$^2$) if the SFQ circuits or JJs are equivalently

[1] An i-line stepper with a wavelength of 365 nm (introduced to the market in the mid-1990s) is used in the fabrication. The state-of-the-art steppers using KrF or ArF excimer lasers would allow the fabrication of ultrafine Josephson junctions and patterns.

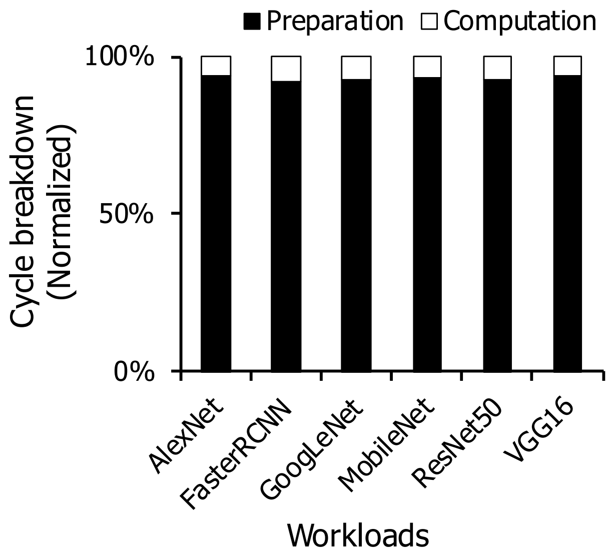


Figure 5.14: Baseline's cycle breakdown normalized for each CNN workload

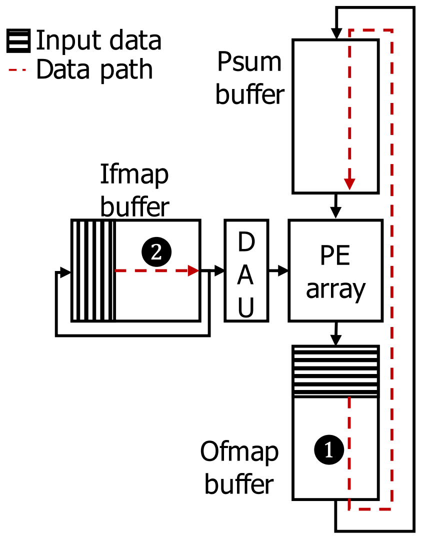


Figure 5.15: Example data path of on-chip buffers

scaled to 28 nm as CMOS technology used in the TPU design[2]. We summarize Baseline's specification including the architectural configurations in Table 5.1.

As Table 5.1 shows, the peak performance of Baseline is significantly high, 3366 TMAC/s, with the clock frequency over 52 GHz. However, we find that the effective performance of Baseline is only about 6.45 TMAC/s on average, which is even lower than 0.2% of its peak performance (Fig. 5.16). In the following subsections, we identify the performance bottlenecks and set the design directions to resolve the identified challenges.

### Bottleneck 1. Huge data movement overhead

We first emphasize the importance of reducing the overhead of data movement among different on-chip buffers and within a single buffer. Fig. 5.14 shows the

[2] To the best of our knowledge, no study mentions the physical limit of JJ scaling. On the other hand, there is the scaling rule that the frequency increases in proportion to the reduction rate of JJ until 200 nm [40], and T-flip-flop (TFF) has successfully demonstrated at up to 770 GHz with the technology [15]. Moreover, there are several schemes to reduce the SFQ cell size without the JJ scaling, such as the introduction of shunt-resistor-free junctions [41], vertically-stacked junctions [13], multi-layer process technology with high-inductance layers [82] and new materials.

Baseline's cycle breakdown normalized for each CNN workload. As the figure clearly indicates, the Baseline's performance is highly dominated by the preparation step (above 90%), which moves data to the appropriate location before starting computation. Based on this analysis, we identify the huge data movement overhead as the first performance bottleneck.

Fig. 5.15 shows the data movement overhead with the example showing the data location right after the end of computation for one weight mapping. First, the calculated partial sums in the ofmap buffer should move to the psum buffer when they need to be accumulated with the next computation result (Fig. 5.15 ❶). In this case, the Baseline should consume a huge amount of cycles corresponding to the sum of two buffers' length, 65,536 cycles (= 16 MB ÷ 256 B/cycle), due to the shift-register-based memory implementation. Also, the ifmap buffer suffers from a similar situation to move the data from its tail to the head when the used ifmap data is required for the next computation again (Fig. 5.15 ❷). Therefore, we conclude that we should minimize the wasteful length of the data movement.

**Bottleneck 2. Fast but idle computing units**

Next, we emphasize the importance of improving the computing unit's (i.e., PE array) utilization. Fig. 5.16 shows the Baseline's roofline plot, which represents the highest achievable performance for a given computational intensity. In this work, we define computational intensity as the number of MAC operations executed with one weight data mapped on the PE. Note that it includes the impact of input batch size on the amount of data reuse.

With the roofline model, Fig. 5.16 shows the performance and computational intensity of each workload with a single input batch. Even though the Baseline's computing units are fast, they are mostly idle with the maximum PE utilization (= roofline performance ÷ peak performance) below 2% on average. The under-utilization directly results from the workloads' low computational intensity and the relatively slow memory access (vs. 52 GHz computation speed). Therefore, we conclude that we should maximize the PE utilization by increasing the computational intensity.

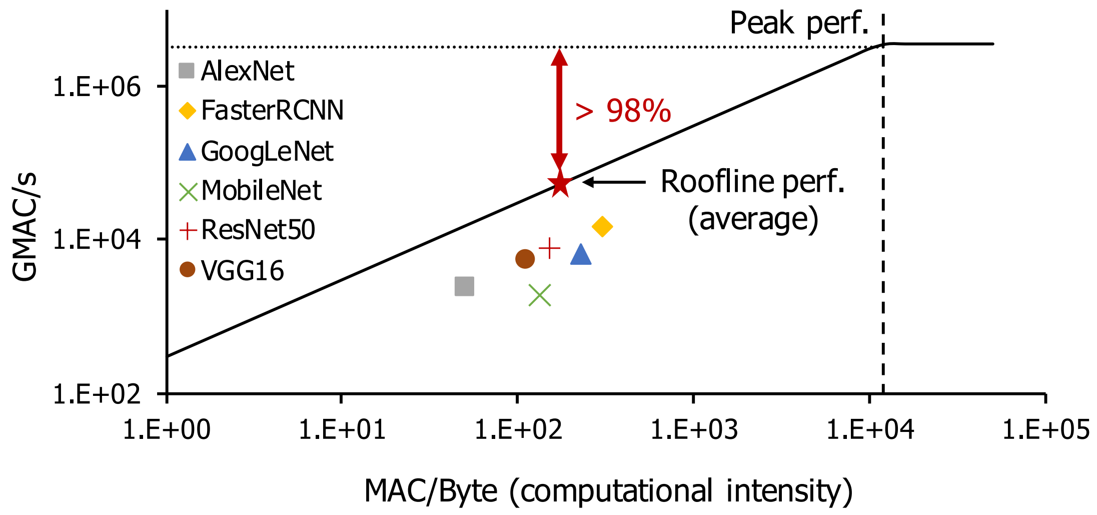
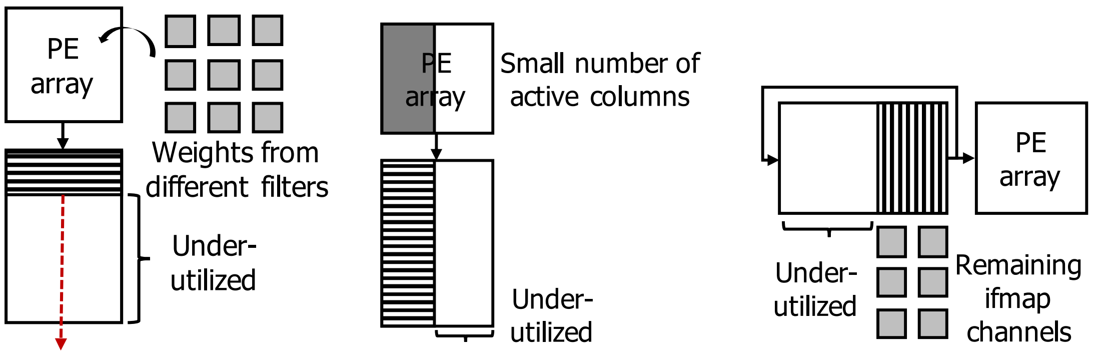


Figure 5.16: Limited performance improvement in Baseline due to the low computational intensity with a single batch

### Bottleneck 3. Waste of on-chip buffer capacity

Lastly, we highlight that it is crucial to resolve the on-chip buffer underutilization issue. To increase the computational intensity, it is required to increase the input batch size for DNN workloads. However, it is highly difficult for the Baseline to take larger batch sizes (i.e., more than one) without additional off-chip memory access because the on-chip buffer can be significantly underutilized.

Fig. 5.17 shows the three scenarios to explain the buffer underutilization problem, which happens even with a single batch. First, even with the huge amount of empty space, the ofmap buffer should flush the data when the next computation is for the different set of output channels (Fig. 5.17(a)). Second, we waste the ofmap buffer's capacity when the number of active PE columns is smaller than the ofmap buffer's width (Fig. 5.17(b)). Third, we cannot map the remaining ifmap channels to the ifmap buffer because each buffer row is dedicated to a given ifmap channel (Fig. 5.17(c)). Therefore, we conclude that we should maximize the buffer utilization by resolving all the mentioned cases.



(a) (b) (c)

Figure 5.17: On-chip buffer under-utilization in terms of (a) ofmap buffer's length, (b) ofmap buffer's width, and (c) ifmap buffer's length

### 5.5.2 SuperNPU: SFQ-optimal NPU architecture

The design implications for optimizing SFQ-based NPU are summarized as follows. First, the optimal design should have a short path for the on-chip buffer data movement. Next, the optimal design should be able to take a large batch size without additional off-chip memory access. Lastly, to achieve the goal, the buffer underutilization problem also should be resolved.

Following all the implications, we design the optimal SFQ-based NPU architecture, *SuperNPU*, as shown in Fig. 5.18. First, SuperNPU has the optimized on-chip buffer architecture where each on-chip buffer is divided into small chunks and connected by the multiplexer and demultiplexer trees. Note that SuperNPU does not have a separated psum buffer but integrates it with the ofmap buffer. Next, the PE array width of SuperNPU is reduced to 1/4, and the on-chip buffer capacity becomes twice compared to the Baseline. Finally, each PE in SuperNPU has eight registers, so a PE can hold eight different weights simultaneously. In the following subsections, we explain how each design choice resolves the identified performance bottlenecks in detail.

#### Optimized on-chip buffer architecture

The optimized on-chip buffer architecture meets the design implications as follows. First, SuperNPU removes the unnecessary data movement and increases the effective buffer capacity by integrating the psum buffer and the ofmap buffer. For

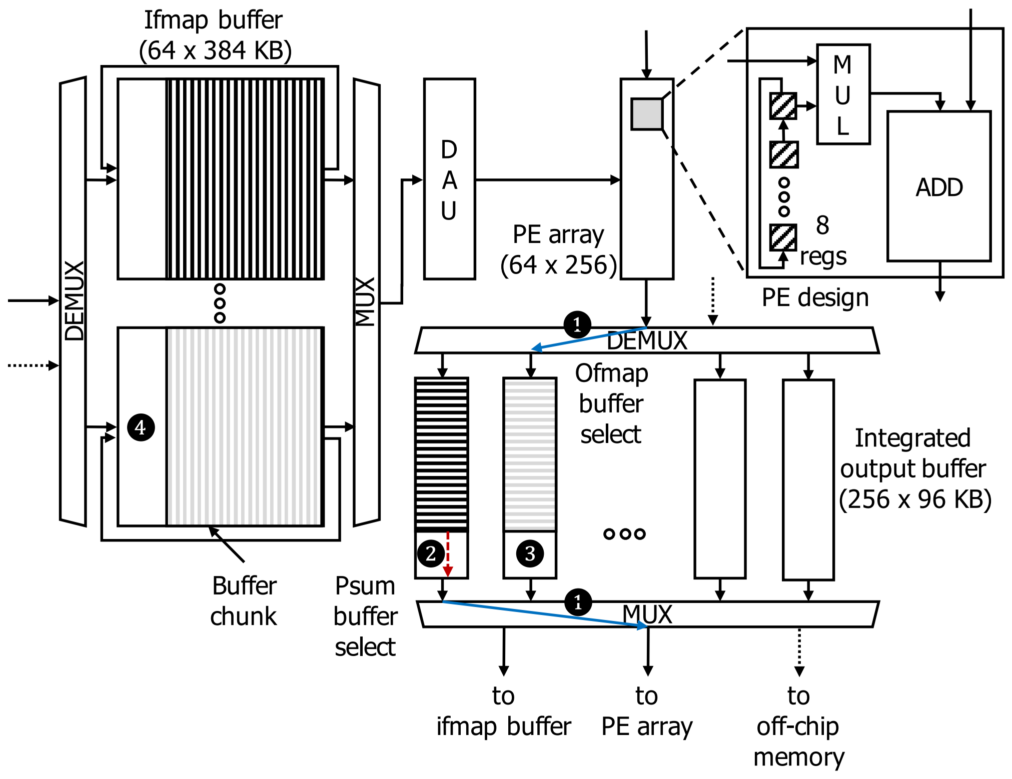


Figure 5.18: SuperNPU overview

example, SuperNPU does not have to move the calculated psum to other buffer chunks. Instead, it just selects the buffer chunk with the psum data as the psum buffer, and one of the empty buffer chunks as the ofmap buffer (Fig. 5.18 ❶). Moreover, by individually selecting the ofmap buffer and psum buffer through the separated multiplexer/decoder, we can flexibly utilize the integrated buffer. Fig. 5.19 shows the performance impact of buffer integration. To match the capacity of input and output buffer, we adjust each buffer's capacity to 12 MB.

Next, by dividing each buffer to several small buffer chunks, SuperNPU significantly reduces data movement overhead in the on-chip buffer (Fig. 5.18 ❷). Our simulation results (Fig. 5.19) show the performance impact of dividing buffer with the various degree of division. With the increasing division degree, the single batch performance continuously increases and achieves 6.26 times higher performance compared to Baseline, from the division degree of 64. Such performance improvement mainly originates from the shortened buffer length, which correpondingly reduces the data-shifting cycles. This result indicates that the buffer division

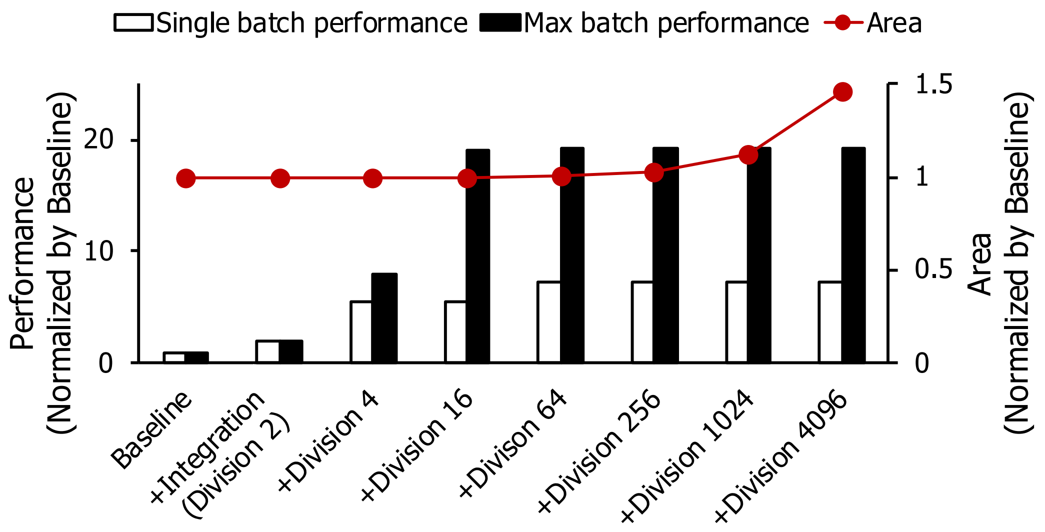


Figure 5.19: Performance impact and area overhead of the buffer optimizations

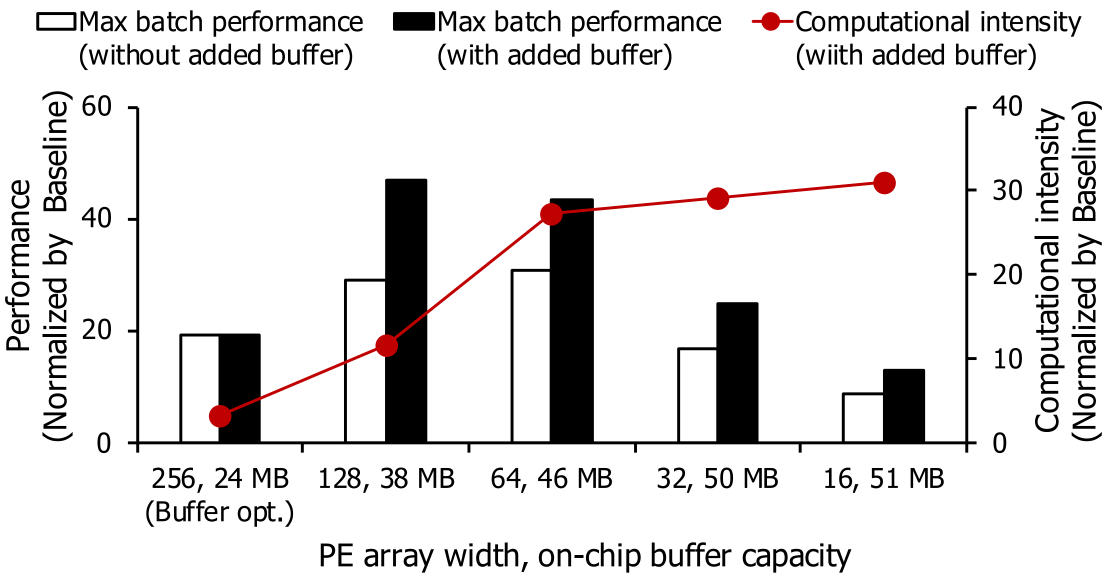


Figure 5.20: Performance and computational intensity with resource balancing

successfully resolves the performance bottleneck resulting form data movement overhead.

The buffer division also mitigates the buffer underutilization issues. For example, the NPU does not need to flush the calculated data in ofmap buffer if there are remaining buffer chunks to hold it, as shown in Fig. 5.18 ❸ (i.e., resolves Fig. 5.17(a)). Moreover, divided ifmap buffer can hold many input channels, corresponding to the PE array height multiplied by the number of buffer chunks in maximum, as shown in Fig. 5.18 ❹ (i.e., resolves Fig. 5.17(c)). Fig. 5.19 shows the impact of improved buffer utilization on the performance with the maximum batch size for each workload. The performance continuously increases and achieves 20 times higher performance from the division degree of 64. Based on the result, we set the buffer division degree as 64 because the performance is saturated. Note that further division incurs the exponentially increasing area overhead of multiplexer/decoder (Fig. 5.19).

**Efficient resource balancing**

Next, in SuperNPU, we increase the on-chip buffer capacity by reducing the number of PEs in the PE array. The insight for this design choice is that there is more room to increase the computational intensity by sacrificing the excessively-high peak performance. Note that we cannot utilize the current peak performance without increasing the computational intensity furthermore (Fig. 5.16).

When reducing the number of PEs, we do not reduce the height of the PE array but its width, to simultaneously resolve the remaining buffer underutilization issue. Even with the optimized on-chip buffer architecture, we still cannot fully utilize the output buffer due to the problem shown in Fig. 5.17b. However, this underutilization issue naturally disappears with the reduced PE array width because the width of the output buffer correspondingly decreases. While reducing the PE array width, we correspondingly divide the integrated output buffer further (i.e., division degree from 64 to 256) to maintain the length of each buffer chunk.

Fig. 5.20 shows the performance impact of resouce balancing which increases the on-chip buffer capacity while reducing the PE array width. In our analysis, we start from the design with the optimized on-chip buffer (256, 24 MB (Buffer

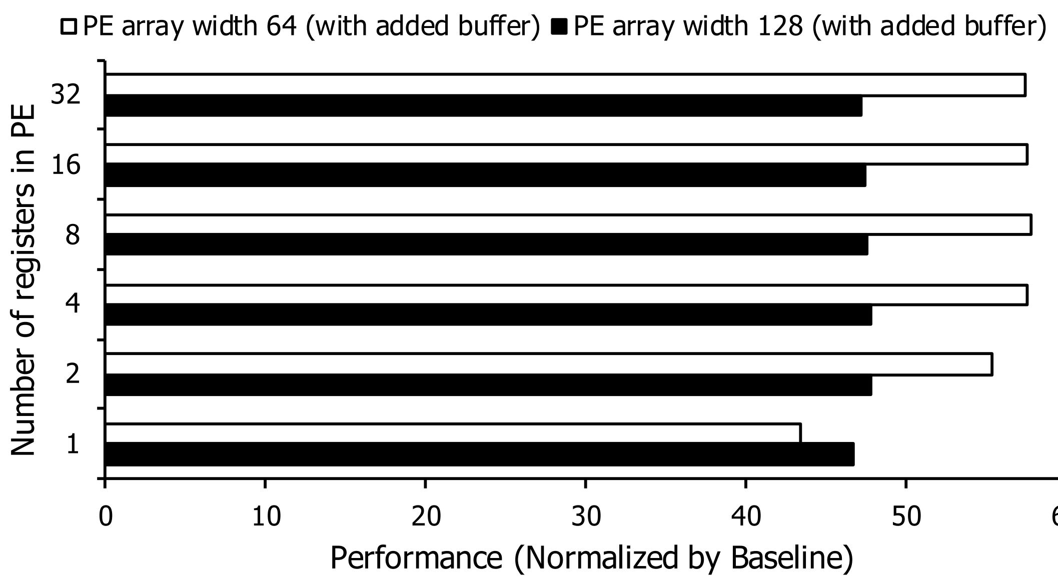


Figure 5.21: Performance impact of number of registers in PE

opt.)). Max batch (without added buffer) indicates that the NPU with the given PE array width and fixed 24 MB on-chip buffer (i.e., no additional capacity). On the other hand, Max batch (with added buffer) has the increased buffer capacity corresponding to the values shown in the graph. We derive each on-chip buffer capacity based on the area occupancy of the PE array and the on-chip buffers.

First, Max batch performance (without added buffer) increases to around 30 times higher compared to Baseline, even though the peak performance decreases. This result indicates that the PE array width reduction itself increases the computational intensity by improving the buffer utilization. Next, with the increased buffer capacity, performance (Max batch performance (with added buffer)) is further improved to 47 times and 42 times higher compared to Baseline in the PE-array width of 128 and 64, respectively. Although the optimal PE array width is 128 in this analysis, the design with the PE array width of 64 has more room for the performance improvement with a much higher computational intensity. Therefore, we focus on these two NPU architectures in the following subsection.

#### Increasing the number of registers in PE

Finally, to further improve the performance, we increase the number of weight registers in PE. With the larger number of weight registers in each PE, SuperNPU increases the PE utilization by filling several PE pipeline stages with a single ifmap data. For example, if each PE holds four different weights from different weight filters, PE can compute four different MAC operations with one ifmap pixel.

Fig. 5.21 shows the performance impact of the number of registers in PE on two chosen designs (128-width and 64-width PE arrays). First, the PE array width of 128 (with added buffer) cannot improve its performance further due to its lower computational intensity, i.e., performance is bounded by the relatively slow memory access. On the other hand, in the PE array width of 64, we get much higher performance improvement thanks to its high computational intensity (Fig. 5.20). Based on this performance analysis, we take the PE array width of 64, the on-chip buffer capacity of 46 MB, and eight registers per PE in SuperNPU.

## 5.6 Evaluation

In this section, we show the system-level performance and power efficiency of SuperNPU by pointing out the impact of each optimization scheme step by step. We first introduce our evaluation methodology (Section 5.6.1). Next, we evaluate SuperNPU in terms of the performance (Section 5.6.2) and performance per Watt (Section 5.6.4).

### 5.6.1 Evaluation methodology

#### Evaluation setup

We evaluate SuperNPU by comparing it with the TPU core [39], one of the most representative server-side DNN accelerators. To estimate the TPU core's performance, we use SCALE-SIM [63], systolic-array-based cycle-accurate DNN accelerator simulator, with the hardware specification summarized in Table 5.1. For the TPU's power consumption, we take 40 W as its average value based on [39]. Also, we set the memory bandwidth of TPU core as 300 GB/s following TPUv2 board

Table 5.1: Evaluation setup

| | TPU | Baseline | Buffer opt. | Resource opt. | Super-NPU |
|---|---|---|---|---|---|
| PE array width | 256 | 256 | 256 | 64 | 64 |
| PE array height | 256 | 256 | 256 | 256 | 256 |
| Ifmap buf. | 24 MB | 8 MB | 12 MB | 24 MB | 24 MB |
| Ofmap buf. | | 8 MB | 12 MB | 24 MB | 24 MB |
| Psum buf. | | 8 MB | | | |
| Weight buf. | | 64 KB | 64 KB | 16 KB | 128 KB |
| # regs in PE | 1 | 1 | 1 | 1 | 8 |
| Frequency (GHz) | 0.7 | 52.6 | 52.6 | 52.6 | 52.6 |
| Peak perf. (TMAC/s) | 45 | 3366 | 3366 | 842 | 842 |
| Area (mm$^2$) (28nm) | <330 | ~283 | ~285 | ~298 | ~299 |

specification [2].

In our performance evaluation, we explicitly show the performance impact of each optimization step by accumulatively evaluating three intermediate SFQ-based NPU architecture designs: architecture introduced in Section 5.5.1 (Baseline), with optimized on-chip buffer (Buffer opt.), and with reduced PE array and larger buffer capacity (Resource opt.). We also set the memory bandwidth for SFQ-based NPU designs as same as the TPU core, 300 GB/s. As the fabrication technology, we take AIST 1.0 $\mu$m process as same as in Section 5.5. We summarize the setup for each architecture in Table 5.1.

For the evaluation, we use six representative CNN workloads that have various application characteristics (e.g., computational intensity, layer configurations). We set each workload's batch size as the maximum value, which can be held by a given on-chip buffer capacity without additional off-chip memory access. For example, for TPU, we set the batch size of AlexNet as 22 because its largest layer's (second layer) input/output data size is 1.05 MB, where 22 input batches can be held within 24 MB in maximum. Our batch setup is conservative because there is

Table 5.2: Workload setup (batch size)

| | TPU | Baseline | Buffer opt. | Resource opt. | Super-NPU |
|---|---|---|---|---|---|
| AlexNet | 22 | 1 | 15 | 30 | 30 |
| FasterRCNN | 20 | 1 | 3 | 30 | 30 |
| GoogLeNet | 20 | 1 | 3 | 30 | 30 |
| MobileNet | 20 | 1 | 3 | 30 | 30 |
| ResNet50 | 20 | 1 | 3 | 30 | 30 |
| VGG16 | 3 | 1 | 1 | 7 | 7 |

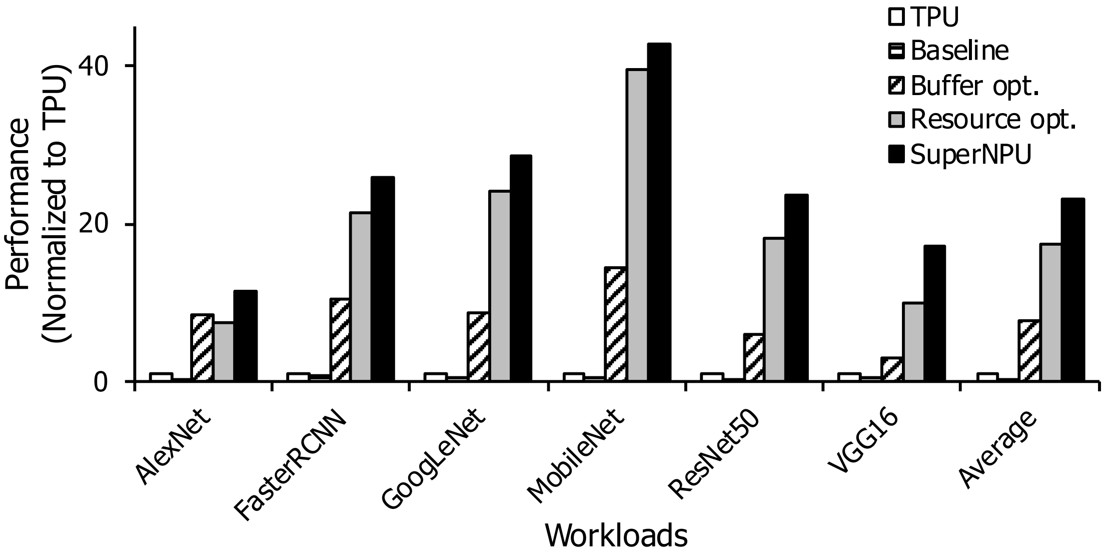


Figure 5.22: Performance evaluation

room to increase the batch size while improving performance. We summarize each workload's batch size setup for all NPU designs in Table 5.2.

## 5.6.2 Performance evaluation

Fig. 5.22 shows the speed-up of SFQ-based NPU designs. The speed-up is calculated by the throughput (i.e., TMAC/s) normalized to that of the TPU. In our performance evaluation, SuperNPU achieves the significant speed-up (23 times) as three architectural optimizations are applied one by one.

The baseline SFQ-based NPU design (Baseline) shows the poor performance, only 40% of TPU's performance on average. The low performance mainly results from the data movement overhead between the on-chip buffers, idle PEs, and low on-chip buffer utilization, as identified in Section 5.5.1. Note that we cannot put even one more input image without off-chip memory overhead in Baseline (Table 5.1).

With the optimized buffer architecture, Buffer opt. achieves the speed-up of 7.7 times on average by resolving the performance bottlenecks in Baseline. The divided and integrated on-chip buffers significantly improve the performance of all workloads by removing the wasteful on-chip buffer data movement. Furthermore, by mitigating the buffer underutilization (Fig. 5.17(a), (c)), most workloads can take benefit of larger batch size (15 in AlexNet and 3 for others except for VGG16).

In Resource opt., average speed-up reaches 17.3 times, mainly thanks to the much higher computational intensity in all workloads. For example, FasterRCNN, GoogLeNet, and MobileNet show the drastically increasing performance in this optimization step with the 10 times larger batch size compared to Buffer opt. Among them, MobileNet shows the highest speed-up (around 40 times) because of its small number of weight filters, usually lower than 64. Note that even with the reduced PE array width, there is no performance degradation in layers consisting of few weight filters. On the other hand, the performance of AlexNet is reduced in Resource opt. due to the reduced peak performance. This is the exact opposite case compared to MobileNet. However, the performance degradation is almost mitigated by the doubled batch size in AlexNet.

Finally, with the increased number of registers in PE, SuperNPU boosts all workloads over 10 times, 23 times on average, and 42 times in MobileNet. In the previous step, the layers consisting of less than 64 filters suffer from performance degradation corresponding to the reduced PE array width. However, in SuperNPU, we mitigate performance degradation by increasing the number of weights in PE (i.e., improving the PE pipeline utilization). For example, in AlexNet, we compensate the performance reduction in Resource opt. by filling the PE pipeline several times with the single input data. As a result, SuperNPU not only successfully shows the performance potential of SFQ-based NPU design but also clearly indicates the importance of optimizing SFQ processors in the right direction.

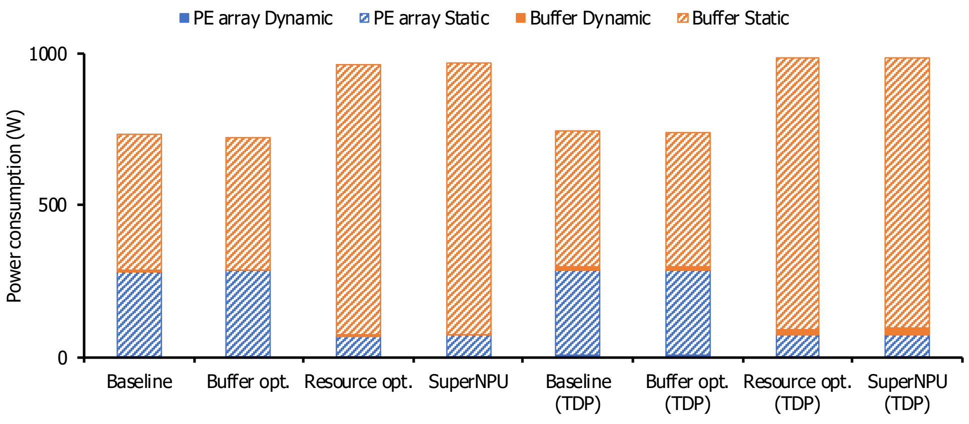


Figure 5.23: Power consumption breakdown of RSFQ-based NPU

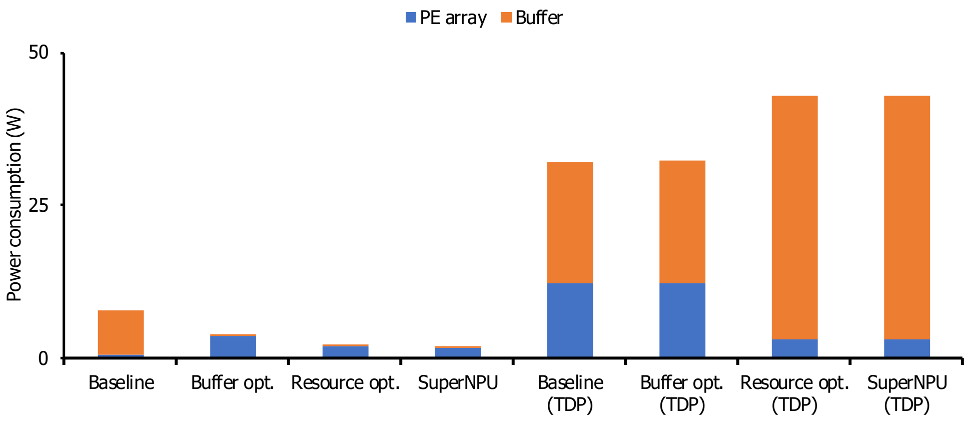


Figure 5.24: Power consumption breakdown of ERSFQ-based NPU

### 5.6.3 Power consumption evaluation

We evaluate SuperNPU's power consumption for two different SFQ device technologies, RSFQ [49], and ERSFQ technology [43]. Fig. 5.23 shows the power consumption breakdown of RSFQ-based NPU. We evaluate both power consumption in CNN execution and the thermal design power (TDP). In the TDP evaluation, we assume that all JJs included in the target NPU switches every cycle. As a result, in the RSFQ-based designs, the static power of both PE array and buffer parts are dominant. This is the reason why many energy-efficient SFQ logic families try to eliminate their static power. Fig. 5.23 also shows the change in the power breakdown between before and after the architectural optimizations. After resource balancing, the buffers' power consumption occupies the majority of total

Table 5.3: Power-efficiency evaluation

| | Power (W) | Performance/W (Normalized to TPU) |
|---|---|---|
| TPU | 40 | 1 |
| RSFQ-SuperNPU (w/o cooling cost) | 964 | 0.95 |
| RSFQ-SuperNPU (w/ cooling cost) | $3.8 \times 10^5$ | 0.002 |
| ERSFQ-SuperNPU (w/o cooling cost) | 1.9 | 490 |
| ERSFQ-SuperNPU (w/ cooling cost) | 751 | 1.23 |

power. Therefore, in the RSFQ logic, the power of the memory part is much larger than that of the computation part if the speed of computation and memory are well balanced.

Fig. 5.24 shows the power consumption breakdown of ERSFQ-based NPU. In contrast to RSFQ power evaluation, the PE array's power consumption occupies most of the total power. This is because the buffer optimization enables NPU to efficiently performs the CNN computation. On the other hand, the result indicates that the baseline seriously suffers from the buffer underutilization and long buffer length, i.e., the long preparing cycles. Moreover, this difference implies that the power-efficiency optimization direction can be totally different in RSFQ and ERSFQ technologies.

### 5.6.4 Power-efficiency evaluation

We evaluate SuperNPU's power-efficiency for two different SFQ device technologies, RSFQ [49], and ERSFQ technology [43]. Table 5.3 shows the power consumption and performance per Watt (i.e., power-efficiency) for SuperNPUs and TPU core. Power-efficiency values are normalized to that of TPU core, which dissipates 40 W in its operation [39]. Also, to include the cooling cost for the 4 K, we set the cooling cost as the 400 times of NPU's power consumption following [34]. Note that the cooling cost is the power consumption of cryocooler for maintaining the target chip at 4 kelvin and does not include the power consumption to build a 4-kelvin environment. In our power-efficiency evaluation, SuperNPU shows 490 times higher power-efficiency provided the free cooling, with ERSFQ technology.

With RSFQ device technology, SuperNPU consumes 964 W, which is infeasible power consumption, due to its huge static power dissipation. Even though its low switching energy, RSFQ technology requires to supply DC-biased current (i.e., DC-biased voltage with bias resistor) for each JJ for the operation (2.5 mV and 70 $\mu$A, respectively). Recently, although several device-level optimizations are proposed to reduce the static power, 960 W is too high to make this technology feasible. As Table 5.3 shows, the power efficiency of RSFQ-SuperNPU is not much lower than TPU (95%) thanks to the 23 times of speed-up. However, with the cooling cost included, the normalized power efficiency value becomes 0.002.

On the other hand, ERSFQ-SuperNPU consumes only 1.9 W because there is no static power consumption in ERSFQ technology [43, 54]. As ERSFQ provides the bias current using JJ with inductors (i.e., bias resistors are replaced to bias JJ), it does not consume static power, but the number of JJs increases (i.e., twice higher dynamic energy per switching). However, thanks to the significantly low switching energy of JJs, ERSFQ-SuperNPU achieves 490 times higher power efficiency compared to the TPU with free cooling provided. Even including the 400 times of cooling cost, ERSFQ-SuperNPU attains 1.23 times higher power efficiency. That is, with ERSFQ-SuperNPU, architects can increase the server-side NPU's performance to 23 times with 490 times higher power-efficiency with assuming free cooling.

## 5.7 Related work

Exploiting emerging devices is a critical challenge to design next-generation computer systems. Many researchers have so far been proposed and discussed such novel architectures. In this section, we discuss prior work from the viewpoint of neural network (NN) acceleration and superconducting computing to clarify the novelty of this paper.

A lot of researchers have proposed NN accelerators for power-efficient processing [17, 4, 29, 61, 48]. A representative approach exploiting an emerging device is to implement memristor-based dot-product operations [67, 18, 7]. Another direction is to introduce PIM (Processor-In-Memory) and die-stacking technologies [50, 18, 85, 42]. A more challenging attempt is to apply nanophotonic technology [28, 69, 44, 74, 68], or superconducting SQUIDs [19, 5] to NN operations. Unlike previous researches, this paper focuses on SFQ circuits and achieves better performance in both the computing power and energy efficiency than the conventional CMOS designs even with the cooling penalty.

Prior researches regarding SFQ demonstrated its significant potential from the viewpoint of circuit implementation. Regardless of its high-speed operations, unfortunately, their throughput was quite low due to the simple but bit-serial designs [87, 6]. Although a recent design successfully demonstrated high-throughout bit-parallel multiplier [55, 36], it is still not clear whether or not the SFQ technology can realize at the system level. Swamit et al. proposed an accelerator for SHA-256 for low latency operations [81]. Tzimpragos et al. introduced an interesting idea that attempts to apply the concept of the delay-based logic (race logic) [52] to SFQ [83]. Another relating proposal is to use not SFQ but AQFP [51] for stochastic computing [12]. Our target is to explore and optimize the architecture of the SFQ-based NPU and to clarify the system-wide potential. To achieve this goal, we have developed a simulation framework, including power/frequency/area models validated based on physical chip fabrication or post-layout characterizations. Also, we have deeply evaluated and presented the significant potential of SFQ devices at the architectural level.

## 5.8 Conclusion

Superconductor SFQ technology is a highly promising solution in post-Moore's era. However, SFQ computing has not yet been realized because of the lack of understanding of SFQ technologies' potentials and limitations. This chapter resolves the challenge as follows. First, we implement and validate an SFQ-based NPU modeling framework. Next, by using the tool, we identify critical challenges in architecting an SFQ-based NPU. Finally, we present SuperNPU, our example SFQ-based NPU architecture, which effectively addresses the challenges at the architectural level. Our evaluation shows that the proposed design outperforms a conventional state-of-the-art NPU by 23 times with comparable power efficiency, even including the extremely expensive cooling costs. We believe that our design methodology can also be applied to architect other SFQ-based architectural units.

This is the first work to show the real potential of SFQ computing, and it focuses on only the inference part of neural networks as a first step. We believe that our work can also be applied to the learning part because the learning part of neural networks also mainly consists of MAC operations. It requires higher precision calculations compared to the inference, such as floating-point calculations. However, there are no floating-point adders or multipliers which employ our proposed architecture design guideline. Therefore, this is the future work to support the learning part of neural networks. Moreover, we only consider the cooling cost for keeping the target chip at 4 kelvin in this chapter. However, it is necessary to consider various other factors such as the communication cost with a conventional CMOS-based computer at room temperature and the SFQ computer's installation location when considering practical use. For SFQ computers to see the light of day in the future, it is necessary to carefully investigate the target application of SFQ computers and evaluate their power performance in consideration of installing these computers and the communication cost with room-temperature computers. We believe that our work is the starting point to realize SFQ-based high-performance computing.

# Chapter 6

# Conclusions

We are currently facing the era where Moore's Law, which has so far contributed to the computer systems' improvement, does not hold anymore. In this era, we are running out of an effective option to improve the performance of the computer system while maintaining its power and temperature budget. This dissertation tackles the problem by using superconductor SFQ technology.

Due to its device's high potentials, many SFQ-related research efforts have been made in various aspects, especially in the device and circuit area. Many physical implementations have successfully demonstrated at the outstanding frequency to show the device potential and feasibility of SFQ logic. However, these designs have prioritized successful demonstrations, and the real potential and effectiveness of SFQ computing are still not clear.

Therefore, this dissertation firstly explores the architectural design space of SFQ processors to maximize the device potential at the system level while minimizing its limitations. As a result, we propose the architectural design guidelines, bit-parallel processing with gate-level deep pipeline structure, are suitable for achieving high performance. This is a novel architecture exploiting its unique natures and entirely different from conventional CMOS technology. Besides, we propose fine-grained multithreading execution as the pipeline stall concealment technologies to prevent significant performance degradation. Although the fine-grained multithreading can keep the circuits simple, the target applications are limited. Moreover, it will be challenging to achieve the peak performance due to the required number of threads increasing in proportion to the number of pipeline stages (i.e., the circuit's scale). Therefore, we must select the appropriate applications suitable for such a deep pipeline nature for achieving high-performance SFQ computing.

Next, this dissertation design and implement a 4-bit SFQ processor based on our proposed architecture to evaluate our proposal's effectiveness and feasibility. As a result of real chip measurement, we confirm the correct operation at 32 GHz with 6.5 mW. The chip consists of 23,713 JJs on $4.1 \times 5.3\text{mm}^2$ area, and this is one of the largest demonstrated circuits. Fortunately, the circuit scale is limited to a few tens of thousands of JJs due to the chip area constraint. However, it becomes quite hard to design SFQ circuits without any design automation tools if the fabrication process technology evolves. For the future development of SFQ computing, it is essential not only to study architecture but also to develop such automation tools. We believe that our work highly motivates industry and academia to work on SFQ design automation technology.

Finally, this dissertation proposes the SFQ-based NPU architecture based on the architectural bottleneck analyses to show the real potential of SFQ computing. Specifically, we implement and validate the SFQ-based NPU simulation framework and optimize the architecture. It is the first work to model and validate a simulator for SFQ-based architectures. Besides, we identify and resolve critical architectural bottlenecks by using the tool and propose the optimized architecture, SuperNPU, that provides extreme performance and power efficiency by outperforming a conventional CMOS design by 23 times and 490 times, respectively. This work focuses on the inference part of neural networks as a first step, and we simply estimate the cooling cost for keeping the target chip at 4 kelvin to evaluate system-level potential. However, it is necessary to consider various other factors such as the communication cost with a conventional CMOS-based computer at room temperature and the SFQ computer's installation location when considering practical use. For SFQ computers to see the light of day in the future, it is necessary to carefully investigate the target application of SFQ computers and evaluate their power performance in consideration of installing these computers and the communication cost with room-temperature computers. We believe that our work is the starting point to realize SFQ-based high-performance computing.

This dissertation focuses on SFQ technology and applying it to processors and accelerators. This is just one application of SFQ technology, and various other applications are possible such as quantum computer's controller or completely different computation logic, e.g., time-domain computing. This technology is still full of wonders and unclear things for us; in other words, SFQ logic has infinite potential. On the other hand, there are still a lot of limitations or constraints for realizing the practical use of SFQ computing, and these issues are summarized as follows.

- **Lack of on-chip and off-chip memory**: SFQ circuits can operate at outstanding speed, such as several tens of GHz even with immature device size. It is essential to build a high-speed memory system that can exploit SFQ circuits' ultra-high-speed nature.
- **Low integration density**: Although SFQ circuits have high-performance potential with current device technology, the device size is quite larger than that of conventional CMOS technology, and it is hard to integrate JJs in a chip.
- **High cooling cost**: Our results clearly shows the expensive cooling cost for keeping the target chip at 4 kelvin, i.e., 490 times higher power efficiency becomes only 1.2 times when considering the cooling cost. There are mainly two ways to suppress the cooling cost: 1) improving the efficiency of cryocooler, 2) high-temperature (e.g., 10 or 30 kelvin) operation with new materials.
- **Interface with room temperature**: This dissertation shows SFQ computing is suitable for the accelerators. In other words, SFQ computers need to communicate with conventional CMOS-based computers at room temperature. The interface can be one of the bottlenecks, and thus, it is essential to develop low-cost interface technology between SFQ and conventional CMOS computers.

Although there are still several challenges for practical use, this dissertation successfully shows the high potential of SFQ computing by device/circuit/architecture level co-designs even with the immature device technology. We believe that our work highly motivates industry and academia to work on SFQ technology to prepare for post-Moore's era. We also believe that our methodology (e.g., exploring and designing the architecture for a novel technology) can also be applied to other emerging technology.

# Acknowledgement

Many teachers, friends, colleagues, and family members have contributed to this dissertation. I am deeply grateful for their contributions.

First and foremost, I want to thank my advisor, Prof. Koji Inoue, for providing me the freedom and resources to pursue my interests, for giving me so many thoughtful and constructive advice on my research, for making an opportunity to go to study abroad. I feel very fortunate to complete a Ph.D. under your supervision.

Besides my advisor, I would like to thank my dissertation committee members, Prof. Kenji Hisazumi and Prof. Yusuke Matsunaga, for their essential comments from the different point of view. These advice from the specialists are irreplaceable for this dissertation.

Next, I would like to thank Prof. Tanaka, who has supported my research for a long time. Thank you for teaching me the fundamentals of research and discussing many research topics with me so often. My research would not have come to be without your dedication and encouragement.

Furthermore, I would like to express my sincere gratitude to Prof. Takatsugu Ono. He managed to find time to attend my meeting and gave me essential comments and advice. He taught not only about the technical things but also what research life to be.

I would like to be grateful to Prof. Jangwoo Kim, who accepted my visit to his laboratory at Seoul University. His advice is essential for my study to progress. I also thank Dr. Ilkwon Byun for his support about my life in Korea and my study to progress.

Special thanks to all the members of the Cyber-Physical Computing Laboratory, including Prof. Teruo Tanimoto, Prof. Satoshi Kawakami, Dr. Keitaro

Oka, Mr. Kosuke Fukumitsu, Ms. Hiroko Fujise, and Ms. Junko Ochiai. I want to thank the Nagoya University members, including Prof. Akira Fujimaki, Prof Taro Yamashita, Dr. Ikki Nagaoka, and Ms. Yasuyo Ogawa. I also thank the Seoul University members, including Mr. Dongmoon Min and Mr. Gyu-Hyeon Lee. Thanks to my dissertation committee members, Prof. Kenji Hisazumi and Prof. Yusuke Matsunaga. To my friends: thank you very much for your love and support.

Finally, I would like to express my deepest gratitude to my family for supporting me and watching over me warmly. I also thank my girlfriend for her encouragement and support.

# List of Publications by the Author

1. Koki Ishida, Masamitsu Tanaka, Takatsugu Ono, and Koji Inoue, "Single-flux-quantum cache memory architecture", 2016 International SoC Design Conference (ISOCC), pp. 105-106, 2016.

2. Koki Ishida, Masamitsu Tanaka, Takatsugu Ono, and Koji Inoue, "Exploring design space of a single-flux-quantum microprocessor (In Japanese)", IPSJ Journal, vol.58, no.3, 2017.

3. Koki Ishida, Masamitsu Tanaka, Ikki Nagaoka, Takatsugu Ono, Satoshi Kawakami, Teruo Tanimoto, Akira Fujimaki, and Koji Inoue, "32 GHz 6.5 mW Gate-Level-Pipelined 4-Bit Processor using Superconductor Single-Flux-Quantum Logic", 2020 IEEE Symposium on VLSI Circuits, pp. 1-2, 2020.

4. Koki Ishida, Ilkwon Byun, Ikki Nagaoka, Kosuke Fukumitsu, Masamitsu Tanaka, Satoshi Kawakami, Teruo Tanimoto, Takatsugu Ono, Jangwoo Kim, and Koji Inoue, "SuperNPU: An Extremely Fast Neural Processing Unit Using Superconducting Logic Devices", 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 58-72, 2020.

# References

[1] 42 Years of Microprocessor Trend Data. https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/.

[2] Hot Chips 2017: A Closer Look At Google's TPU v2. https://www.tomshardware.com/news/tpu-v2-google-machine-learning,35370.html.

[3] Intel's Core i7 processors. https://techreport.com/review/15818/intels-core-i7-processors/.

[4] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. *ACM SIGARCH Computer Architecture News*, 44(3):1–13, 2016.

[5] M. Altay Karamuftuoglu and Ali Bozbey. Single Flux Quantum Based Ultra-high Speed Spiking Neuron. *arXiv e-prints*, page arXiv:1812.10354, December 2018.

[6] 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(5):1–5, 2016.

[7] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojicic. PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the 24th International Conference on Architectural Support for*

*Programming Languages and Operating Systems*, ASPLOS ’19, page 715–731, New York, NY, USA, 2019. Association for Computing Machinery.

[8] W. Aspray. The intel 4004 microprocessor: what constituted invention? *IEEE Annals of the History of Computing*, 19(3):4–15, 1997.

[9] Manjul Bhushan, Paul Bunyk, Michael Cuthbert, Erik P. DeBenedictis, Michael Frank, and Travis Humble. Cryogenic electronics and quantum information processing. 6 2019.

[10] P. Bunyk and P. Litskevitch. Case study in rsfq design: fast pipelined parallel adder. *IEEE Transactions on Applied Superconductivity*, 9(2):3714–3720, June 1999.

[11] I. Byun, D. Min, G. Lee, S. Na, and J. Kim. Cryocore: A fast and dense processor architecture for cryogenic computing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 335–348, 2020.

[12] Ruizhe Cai, Ao Ren, Olivia Chen, Ning Liu, Caiwen Ding, Xuehai Qian, Jie Han, Wenhui Luo, Nobuyuki Yoshikawa, and Yanzhi 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, page 567–578, New York, NY, USA, 2019. Association for Computing Machinery.

[13] M. A. Castellanos-Beltran, D. I. Olaya, A. J. Sirois, P. D. Dresselhaus, S. P. Benz, and P. F. Hopkins. Stacked Josephson junctions as inductors for single flux quantum circuits. *IEEE Transactions on Applied Superconductivity*, 29(5):1–5, 2019.

[14] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata. Cell broadband engine architecture and its first implementation: A performance view. *IBM J. Res. Dev.*, 51(5):559–572, September 2007.

[15] Wei Chen, Alexander Rylyakov, V. Patel, J.E. Lukens, and K.K. Likharev.

Rapid single flux quantum t-flip flop operating up to 770 GHz. *Applied Superconductivity, IEEE Transactions on*, 9:3212 – 3215, 07 1999.

[16] Y. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016.

[17] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.

[18] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, page 27–39. IEEE Press, 2016.

[19] F Chiarello, P Carelli, M G Castellano, and G Torrioli. Artificial neural network based on SQUIDs: demonstration of network training and operation. *Superconductor Science and Technology*, 26(12):125009, oct 2013.

[20] R. Courtland. Google aims for quantum computing supremacy [news]. *IEEE Spectrum*, 54(6):9–10, 2017.

[21] I. M. Dayton, T. Sage, E. C. Gingrich, M. G. Loving, T. F. Ambrose, N. P. Siwak, S. Keebaugh, C. Kirby, D. L. Miller, A. Y. Herr, Q. P. Herr, and O. Naaman. Experimental demonstration of a Josephson magnetic memory cell with a programmable $\pi$-junction. *IEEE Magnetics Letters*, 9:1–5, 2018.

[22] R. H. Dennard, F. H. Gaensslen, H. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[23] 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(1):326–332, 2001.

[24] M. Dorojevets, Z. Chen, C. L. Ayala, and A. K. Kasperek. Towards 32-bit energy-efficient superconductor RQL processors: The cell-level design and analysis of key processing and on-chip storage units. *IEEE Transactions on Applied Superconductivity*, 25(3):1–8, 2015.

[25] E.S. Fang and T. Van Duzer. A Josephson integrated circuit simulator (JSIM) for superconductive electronics application. *Extended Abstracts of 1989 International Superconductivity Electronics Conference*, pages 407–410, 1989.

[26] B. Flachs, S. Asano, S. H. Dhong, H. P. Hofstee, G. Gervais, Roy Kim, T. Le, Peichun Liu, J. Leenstra, J. Liberty, B. Michael, Hwa-Joon Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, N. Yano, D. A. Brokenshire, M. Peyravian, Vandung To, and E. Iwata. The microarchitecture of the synergistic processor for a cell processor. *IEEE Journal of Solid-State Circuits*, 41(1):63–70, Jan 2006.

[27] L. Gomes. Quantum computing: Both here and not here. *IEEE Spectrum*, 55(4):42–47, 2018.

[28] Matthias Gruber, Jürgen Jahns, and Stefan Sinzinger. Planar-integrated optical vector-matrix multiplier. *Appl. Opt.*, 39(29):5367–5373, Oct 2000.

[29] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[30] R. Harris, M. W. Johnson, T. Lanting, A. J. Berkley, J. Johansson, P. Bunyk, E. Tolkacheva, E. Ladizinsky, N. Ladizinsky, T. Oh, F. Cioata, I. Perminov, P. Spear, C. Enderud, C. Rich, S. Uchaikin, M. C. Thom, E. M. Chapple, J. Wang, B. Wilson, M. H. S. Amin, N. Dickson, K. Karimi, B. Macready, C. J. S. Truncik, and G. Rose. Experimental investigation of an eight-qubit unit cell in a superconducting optimization processor. *Phys. Rev. B*, 82:024511, Jul 2010.

[31] Allan Hartstein and Thomas R Puzak. The optimum pipeline depth for a microprocessor. In *ACM SIGARCH Computer Architecture News*, volume 30, pages 7–13. IEEE Computer Society, 2002.

[32] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[33] Quentin P. Herr, Anna Y. Herr, Oliver T. Oberg, and Alexander G. Ioannidis. Ultra-low-power superconductor logic. *Journal of Applied Physics*, 109(10):103903–103903–8, May 2011.

[34] D. S. Holmes, A. L. Ripple, and M. A. Manheimer. Energy-efficient superconducting computing—power budgets and requirements. *IEEE Transactions on Applied Superconductivity*, 23(3):1701610–1701610, 2013.

[35] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv e-prints*, page arXiv:1704.04861, April 2017.

[36] 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*, pages 1–2, 2020.

[37] K. Ishida, M. Tanaka, T. Ono, and K. Inoue. Single-flux-quantum cache memory architecture. In *2016 International SoC Design Conference (ISOCC)*, pages 105–106, 2016.

[38] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[39] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, page 1–12, New York, NY, USA, 2017. Association for Computing Machinery.

[40] Alan M Kadin, Cesar A Mancini, Marc J Feldman, and Darren K Brock. Can RSFQ logic circuits be scaled to deep submicron junctions? *Applied Superconductivity, IEEE Transactions on*, 11(1):1050–1055, 2001.

[41] Ryohei Kanada, Yuki Nagai, Hiroyuki Akaike, and Akira Fujimaki. Self-Shunted NbN Junctions With $NbN_x/AlN$ Bilayered Barriers for 4 K Operation. *IEEE Transactions on Applied Superconductivity*, 19(3):249–252, June 2009.

[42] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with High-Density 3D memory. *SIGARCH Comput. Archit. News*, 44(3):380–392, June 2016.

[43] D. E. Kirichenko, S. Sarwana, and A. F. Kirichenko. Zero static power dis-

sipation biasing of RSFQ circuits. *IEEE Transactions on Applied Superconductivity*, 21(3):776–779, June 2011.

[44] Ken'ichi Kitayama, Masaya Notomi, Makoto Naruse, Koji Inoue, Satoshi Kawakami, and Atsushi Uchida. Novel frontier of photonics for data processing–photonic accelerator. *APL Photonics*, 4(9):090901, 2019.

[45] G. Konno, Y. Yamanashi, and N. Yoshikawa. Fully functional operation of low-power 64-kb Josephson-CMOS hybrid memories. *IEEE Transactions on Applied Superconductivity*, 27(4):1–7, 2017.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[47] Gyu-hyeon Lee, Dongmoon Min, Ilkwon Byun, and Jangwoo Kim. Cryogenic computer architecture modeling with memory-side case studies. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 774–787, New York, NY, USA, 2019. Association for Computing Machinery.

[48] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. *ACM SIGARCH Computer Architecture News*, 44(3):255–266, 2016.

[49] 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(1):3–28, March 1991.

[50] Jiawen Liu, Hengyu Zhao, Matheus Almeida Ogleari, Dong Li, and Jishen Zhao. Processing-in-Memory for energy-efficient neural network training: A heterogeneous approach. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, page 655–668. IEEE Press, 2018.

[51] K. Loe and E. Goto. Analysis of flux input and output josephson pair device. *IEEE Transactions on Magnetics*, 21(2):884–887, 1985.

[52] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. Race logic: A hardware acceleration for dynamic programming algorithms. *SIGARCH Comput. Archit. News*, 42(3):517–528, June 2014.

[53] Dongmoon Min, Ilkwon Byun, Gyu-Hyeon Lee, Seongmin Na, and Jangwoo Kim. Cryocache: A fast, large, and cost-effective cache architecture for cryogenic computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 449–464, New York, NY, USA, 2020. Association for Computing Machinery.

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

[55] I. Nagaoka, M. Tanaka, K. Inoue, and A. Fujimaki. A 48GHz 5.6mW gate-level-pipelined multiplier using single-flux quantum logic. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 460–462, 2019.

[56] 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)*, pages 1–3, 2019.

[57] Shuichi Nagasawa, Kenji Hinode, Tetsuro Satoh, Mutsuo Hidaka, Hiroyuki Akaike, Akira Fujimaki, Nobuyuki Yoshikawa, Kazuyoshi Takagi, and Naofumi Takagi. Nb 9-layer fabrication process for superconducting large-scale SFQ circuits and its process evaluation. *IEICE Transactions on Electronics*, E97.C(3):132–140, 2014.

[58] K. Nakajima, Y. Onodera, and Y. Ogawa. Logic design of Josephson network. *Journal of Applied Physics*, 47(4):1620–1627, 1976.

[59] Ghasem Pasandi, Alireza Shafaei, and Massoud Pedram. SFQmap: A Technology Mapping Tool for Single Flux Quantum Logic Circuits. *arXiv e-prints*, page arXiv:1901.00894, January 2019.

[60] M. Pedram and Y. Wang. Design automation methodology and tools for superconductive electronics. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6, 2018.

[61] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 267–278. IEEE, 2016.

[62] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv e-prints*, page arXiv:1506.01497, June 2015.

[63] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv e-prints*, page arXiv:1811.02883, October 2018.

[64] R. Sato, Y. Hatanaka, Y. Ando, M. Tanaka, A. Fujimaki, K. Takagi, and N. Takagi. High-speed operation of random-access-memory-embedded microprocessor with minimal instruction set architecture based on rapid single-flux-quantum logic. *IEEE Transactions on Applied Superconductivity*, 27(4):1–5, June 2017.

[65] R. R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, 1997.

[66] E. A. Sete, W. J. Zeng, and C. T. Rigetti. A functional architecture for scalable quantum computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–6, 2016.

[67] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar.

ISAAC: A convolutional neural network accelerator with in-Situ analog arithmetic in crossbars. *SIGARCH Comput. Archit. News*, 44(3):14–26, June 2016.

[68] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. Deep learning with coherent nanophotonic circuits. *Nature Photonics*, 2017.

[69] Kyle Shiflett, Dylan Wright, Avinash Karanth, and Ahmed Louri. PIXEL: Photonic neural network accelerator. In *Proceedings of the 26th IEEE International Symposium on High-Performance Computer Architecture*, HPCA '20, February 2020.

[70] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[71] Eric Sprangle and Doug Carmean. Increasing processor performance by implementing deeper pipelines. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 25–34. IEEE, 2002.

[72] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv e-prints*, page arXiv:1409.4842, September 2014.

[73] Shuichi Tahara, Ichiro Ishida, Yumi Ajisawa, and Yoshifusa Wada. Experimental vortex transitional nondestructive read-out Josephson memory cell. *Journal of Applied Physics*, 65(2):851–856, January 1989.

[74] Alexander N. Tait, Thomas Ferreira de Lima, Ellen Zhou, Allie X. Wu, Mitchell A. Nahmias, Bhavin J. Shastri, and Paul R. Prucnal. Neuromorphic photonic networks using silicon photonic weight banks. *Scientific Reports*, 7(1), Aug 2017.

[75] K. Takagi, M. Tanaka, S. Iwasaki, R. Kasagi, I. Kataeva, S. Nagasawa, T. Satoh, H. Akaike, and A. Fujimaki. SFQ propagation properties in passive

transmission lines based on a 10-Nb-layer structure. *IEEE Transactions on Applied Superconductivity*, 19(3):617–620, 2009.

[76] M. Tanaka, M. Suzuki, G. Konno, Y. Ito, A. Fujimaki, and N. Yoshikawa. Josephson-CMOS hybrid memory with nanocryotrons. *IEEE Transactions on Applied Superconductivity*, 27(4):1–4, 2017.

[77] M Tanaka, Y Yamanashi, N Irie, HJ Park, S Iwasaki, K Takagi, K Taketomi, A Fujimaki, N Yoshikawa, H Terai, et al. Design and implementation of a pipelined 8 bit-serial single-flux-quantum microprocessor with cache memories. *Superconductor Science and Technology*, 20(11):S305–S309, 2007.

[78] Masamitsu Tanaka, Masato Ito, Atsushi Kitayama, Tomohito Kouketsu, and Akira Fujimaki. 18-GHz, 4.0-aJ/bit operation of ultra-low-energy rapid single-flux-quantum shift registers. *Japanese Journal of Applied Physics*, 51:053102, may 2012.

[79] Masamitsu Tanaka, F. Matsuzaki, T. Kondo, N. Nakajima, Yuki Yamanashi, A. Fujimaki, H. Hayakawa, Nobuyuki Yoshikawa, H. Terai, and S. Yorozu. A single-flux-quantum logic prototype microprocessor. volume 47, pages 298 – 529 Vol.1, 03 2004.

[80] Guang-Ming Tang, Pei-Yao Qu, Xiao-Chun Ye, and Dong-Rui Fan. Logic design of a 16-bit bit-slice arithmetic logic unit for 32-/64-bit RSFQ microprocessors. *IEEE Transactions on Applied Superconductivity*, PP:1–1, 01 2018.

[81] Swamit S. Tannu, Poulami Das, Michael L. Lewis, Robert Krick, Douglas M. Carmean, and Moinuddin K. Qureshi. A case for superconducting accelerators. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, CF '19, page 67–75, New York, NY, USA, 2019. Association for Computing Machinery.

[82] 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(3):1–10, 2016.

[83] Georgios Tzimpragos, Dilip Vasudevan, Nestan Tsiskaridze, George Michelogiannakis, Advait Madhavan, Jennifer Volk, John Shalf, and Timothy Sherwood. A computational temporal logic for superconducting accelerators. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 435–448, New York, NY, USA, 2020. Association for Computing Machinery.

[84] M H Volkmann, A Sahu, C J Fourie, and O A Mukhanov. Implementation of energy efficient single flux quantum digital circuits with sub-aJ/bit operation. *Superconductor Science and Technology*, 26(1):015002, 2013.

[85] P. Wang, Y. Ji, C. Hong, Y. Lyu, D. Wang, and Y. Xie. SNrram: An efficient sparse neural network computation architecture based on resistive random-access memory. In *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[86] Y. Yamanashi, T. Nishigai, and N. Yoshikawa. Study of LR-loading technique for low-power single flux quantum circuits. *IEEE Transactions on Applied Superconductivity*, 17(2):150–153, June 2007.

[87] 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, CORE 1$\beta$. *IEEE Transactions on Applied Superconductivity*, 17(2):474–477, 2007.

[88] Yuki Yamanashi, Toshiki Kainuma, Nobuyuki Yoshikawa, Irina Kataeva, Hiroyuki Akaike, Akira Fujimaki, Masamitsu Tanaka, Naofumi Takagi, Shuichi Nagasawa, and Mutsuo Hidaka. 100 GHz demonstrations based on the single-flux-quantum cell library for the 10 kA/cm$^2$ Nb multi-layer process. *IEICE Transactions on Electronics*, 93(4):440–444, apr 2010.

[89] N Yoshikawa and Y Kato. Reduction of power consumption of RSFQ circuits by inductance-load biasing. *Superconductor Science and Technology*, 12(11):918–920, nov 1999.