2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2027-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPCA53966.2022.00038

HiPerRF: A Dual-Bit Dense Storage SFQ Register File

Haipeng Zha*, Naveen Kumar Katam†, Massoud Pedram*, Murali Annavaram*
*Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, USA
{hzha, pedram, annavara}@usc.edu
†SeeQC Inc.
Elmsford, USA
naveen@seeqc.com

***Abstract*—Single Flux Quantum (SFQ) superconducting technology provides significant power and performance benefits in the era of diminishing CMOS scaling. Recent advances in design tools and fabrication facilities have brought SFQ based computing to the forefront. One challenge faced by SFQ technology is to have a compact and robust on-chip memory, which can be used for implementing register files and cache memory. While dense memories are being investigated through the development of three-terminal devices such as Nanocryotrons, in this work, we build on a novel memory cell built using traditional Josephson junctions (JJs). In particular, we design a high capacity register file, called HiPerRF, that builds on a High Capacity Destructive ReadOut (HC-DRO) cell in SFQ technology. HC-DRO design can store up to three fluxon pulses, thereby providing the equivalent of 2-bit storage in a single cell. However, these cells provide only destructive readout capability, namely each value can be read only once. However, CPU register file contents are read multiple times in any program, and hence a destructive readout complicates register file design. HiPerRF provides the non-destructive property using a loopback write mechanism, thereby preserving the higher density of HC-DRO cells without compromising the multi-read demands of a register file. HiPerRF reduces the JJ count of the register file design, after accounting for all the peripheral access circuitry costs, by 56.1% and reduces the static power by 46.2%. Furthermore, HiPerRF reduces the JJ count by 16.3% even when considering an entire in-order RISC-V CPU core.**

***Keywords*-Superconducting electronics; SFQ; Register File; Destructive Readout;**

I. Introduction

Rapid Single Flux Quantum (RSFQ) devices introduced by Likharev et al. [1] have gained traction as one of the promising technologies to augment CMOS based computing. Single Flux Quantum (SFQ) technology uses quantized voltage pulses in digital data generation, reproduction, amplification, memorization, and processing. In particular, applications and kernels that demand substantial compute density and/or need to operate at extremely low power are well suited for SFQ based computing. Some examples are computing in space applications with extremely limited power and iterative linear algebraic computations in machine learning. The SFQ technology is based on superconducting devices called Josephson Junctions (JJs). These devices work at a low temperature with a short switching time ($\sim$1ps) and little switching energy dissipation ($\sim 10^{-19}$Joules) [2]. The JJ-based SFQ circuits designed have been demonstrated to operate at frequencies up to 770GHz [3]. Recent works focused on efficient SFQ logic circuit realizations, such as designing ALUs, and other digital structures that are necessary to build CPUs [4]–[6]. Even the early realization of a simple 8-bit bit-serial CPU has been prototyped [7]. The current SFQ technology is roughly equivalent to a "250 nm" CMOS node. The SCE technology road map [8] predicts that by 2026 we will have a "90 nm" equivalent node. Theoretical estimations of the maximum density of SFQ-based circuits utilizing the geometric inductance of a wire suggest a density of approximately 10^7 JJ/cm^2. These projections indicate that building a SFQ based CPU is within reasonable reach.

SFQ memories, however, are built as flip-flop-like designs, and hence memory density is quite low compared to SRAM. SFQ provides two different memory cell designs currently: a destructive readout (DRO) cell and a non-destructive readout (NDRO) cell (design details in the next section). Each DRO or NDRO cell in current designs stores a single pulse. To increase the memory capacity, recently, our group proposed a High Capacity Destructive ReadOut (HC-DRO) memory cell [9]. HC-DRO can hold up to three SFQ pulses, which means they can store 2 bits of information in one memory cell, thereby providing an opportunity to double the memory density.

In this paper our goal is to build a CPU register file using HC-DRO cells. We must, however, tackle multiple challenges. The inability to retain data after a single *read* makes HC-DRO (or any DRO) cells challenging for register files. The use of NDRO cells is quite expensive in terms of JJ counts (7X more JJs needed for 2-bit NDRO compared to a single HC-DRO cell). Given that the size of the physical register file has a significant performance impact [10], we propose a solution that relies on HC-DRO cells for high density while at the same time supporting the need for reading each register multiple times. Our design is based on the intuition that only a few registers are actively read at any

978-1-6654-2027-3/22/$31.00 ©2022 IEEE
DOI 10.1109/HPCA53966.2022.00038

given time window. Hence, one needs to preserve the non-destructive read property only for those active registers. As such, we augment a large HC-DRO register file with a small victim NDRO buffer that helps the readout data to recycle back to the original register in a lazy manner, outside of the critical path, thereby providing the non-destructive read property. Prior work [11] has used a rotating shift register where the value of each bit is pushed back from the tail of a shift register to the head. But this work is centered on designing a rotating shift register and does not address the architectural challenges of incorporating destructive readout cells into an architecturally feasible register file design. Given that the architectural challenges of building a register file in a CPU pipeline are substantial, we design HiPerRF to tackle the numerous challenges. This work will discuss these challenges and propose solutions that allow SFQ based CPU designs to exploit HC-DRO cells for register files. The primary contributions of this paper are as follows:

- We present the design of HiPerRF, which is a register file built with HC-DRO cells. We present the design enhancements to read and write multiple SFQ pulses without perturbing the rest of the SFQ CPU pipeline. Since HC-DRO cells lose data after each read, we present an approach to restore the value after each read. We use a set of NDRO cells shared across an entire column of DRO cells to enable a low-cost approach to preserve the HC-DRO cell contents.
- We present a dual-banked design of HiPerRF. The dual-banked HiPerRF accompanied with a static scheduling algorithm reduces the port contention and increases the performance.
- We implemented the design using detailed cell level libraries and a hybrid pipeline-gate level simulation to evaluate the area and power impacts. Given that JJ counts are the primary design limiter in SFQ CPUs, we quantified the JJ reduction benefits of HiPerRF when integrated into an in-order RISC-V CPU. Our results also show that when considering the whole CPU, the HiPerRF design provides a 16.3% reduction in the JJ count.

II. Background

A. SFQ Logic

Unlike the CMOS technology that uses high and low voltage levels to represent ”1” and ”0”, SFQ logic use magnetic pulses. The SFQ pulse is stored in the form of a single quantum flux or fluxon. These fluxons are transmitted between logic gates to enable computations. For memory cells, the existence of an SFQ pulse represents a ”1”, and the absence of a pulse represents a ”0”. However, when performing computations in an SFQ logic gate, the lack of a pulse at the input leads to ambiguity. The lack of a pulse can mean a ”0”, or the pulse has not yet arrived. To disambiguate these two scenarios, SFQ logic uses gate-level clocking. At the end of a clock period, the absence of a pulse at the gate input will be treated as an input ”0” for computation. The use of gate-level clocking is thus unique to SFQ logic, as opposed to pipeline stage-level clocking used in traditional microprocessors.

Since all SFQ logic gates have a clock, the output pulses generated by a logic gate are driven to the next gate in the computation in the next cycle. This characteristic makes the SFQ circuits pipelined down at the gate level. The gate-level clocking requirement leads to deep pipelines, which create data and control hazards more frequently than in CMOS designs. We describe such hazards in the HiPerRF design and present solutions that consider gate-level pipelining impacts. For instance, our design considers various data and control hazards to make sure no two signals arrive at any gate at the same time.

B. Clock Distribution

Gate pipelined circuits may pose a challenge for clock distribution. Prior approaches to tackle clocking challenges include using a hierarchical clock design where a system clock (called the slow clock) is used for advancing functional pipeline stages, such as issue and decode stage, and using local clock (fast clock) for bit-serial operations [12]. Such hierarchical clock distribution solutions help meet the timing requirement and simplify clock design. These are also concurrent advances in clock distribution strategies, which are outside the scope of this paper. For instance, using the dynamic SFQ (DSFQ) technology [13], researchers have designed the gate with self-resetting property, thereby removing the need for a clock to operate each gate. This paper eschews the clock distribution concern by designing a CPU register file architecture that does not need explicit clock distribution for each gate. Instead, the read and write enable signals of the register file (generated from a decoder) automatically activate individual gates in the register access ports to trigger data movement. By using a clock-follow-data like approach, one can avoid the clock distribution demands of the register file design, as we demonstrate in this paper.

C. DRO Memory Cell

Destructive read-out (DRO) cell [1] of single flux quantum (SFQ) technology is one of the most important building blocks for superconducting circuits, which can be used as a memory element for storing the SFQ pulses. It is also used as a buffer cell for synchronizing the signals [14] in a circuit. DRO cell is also known sometimes as an RS-Flipflop or D-Flipflop.

Figure 1(a) shows the schematic of a regular DRO cell which receives an SFQ pulse at *input D* and stores it in the superconducting loop J_1-L_2-J_2 if it does not already have a fluxon (SFQ pulse) stored in it. If the J_1-L_1-J_2 loop already has a fluxon stored in it, the incoming pulse is

dissipated through the buffer junction J_0. The stored fluxon is read by *input CLK* which resets the superconducting loop, subsequently resulting in an SFQ pulse at the *output Q*. Each cell read is destructive since the loop is reset after each read operation. Thus a DRO cell stores at most a single fluxon acting as a one-bit storage cell, and provides destructive read out capability.

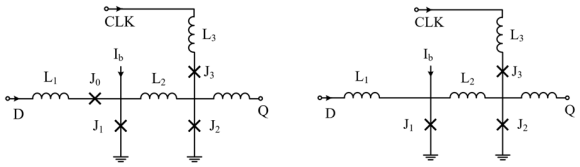

Figure 1. Schematic of (a) DRO Cell (b) HC-DRO cell

D. HC-DRO Memory Cell

In our prior work, we proposed high capacity destructive read-out (HC-DRO) cell [9]. This design is based on an important observation that it is possible to store more than one fluxon in a memory cell, creating the effect of a multi-bit storage cell. Figure 1(b) shows the schematic of the HC-DRO cell. Compared with Figure 1(a), the J_0 JJ has been removed, thereby allowing the accumulation of more than one incoming pulse. The J_1-L_2-J_2 loop can hold multiple pulses by increasing the L_2 inductance. The critical currents of J_1 and J_2 are generally increased to enable stable reading of the multiple pulses.

While storing multiple bits in a storage cell may increase device variability, with careful inductor sizing and critical current delivery to JJs, a 2-bit HC-DRO can be robustly built [9]. We have designed and verified the operation of a robust 2-bit HC-DRO cell using JoSim, a detailed device and circuit simulator for superconducting designs [15]. The design parameters that provided the robust behavior in our design are L1$\sim$ 6 pH, L2$\sim$ 20 pH, L3$\sim$ 4 pH, J1$\sim$ 115 μA, J2$\sim$ 111 μA, J3$\sim$ 80 μA. While device variability may be another concern, we believe the advances in fabrication technology will enable HC-DRO to be built just as robustly as any device.

E. NDRO Memory Cell

Non-destructive read-out (NDRO) cell [16] is another important memory cell in SFQ technology. Unlike DRO cells, NDRO cells can keep the data after the read operation. It works similar to a CMOS D flip-flop with reset. Figure 2 shows the schematic of a regular NDRO cell. Once it receives a pulse from the *input IN*, it will store the fluxon in the loop J_3-L_5-J_7-J_{10}. If the J_3-L_5-J_7-J_{10} loop already has a fluxon stored in it, the incoming *IN* pulse is dissipated through the junction J_2. The pulse that comes from the *input RESET* will make the fluxon stored in the loop to be dissipated through J_7. If there is no pulse stored in the loop, the *RESET* pulse is dissipated through J_5. The pulse from the *input CLK*, which is essentially a read operation, will trigger a pulse on the *output OUT* only if the J_3-L_5-J_7-J_{10} loop has a fluxon, and the stored fluxon stays as is. Thus the NDRO cell keeps the read operation non-destructive. However, this design cost 11 JJs. To store 2-bits with NDRO requires 2×11 JJs. On the other hand, HC-DRO uses only 3 JJs to store 2-bit, hence providing a 7.3× density advantage. Note that in SFQ designs, density is mainly measured in terms of JJ count since JJs are the most critical manufacturing bottleneck.

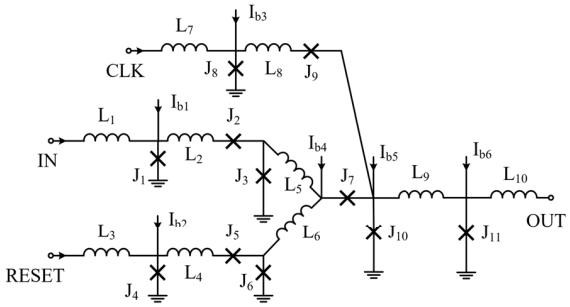

Figure 2. Schematic of an NDRO Cell

F. Splitters and Mergers

Because a single pulse is generated from a logic gate, it is not possible to drive two SFQ gates by one SFQ pulse. Unlike CMOS fan-out junctions, an SFQ pulse must be explicitly split at every fan-out point. Thus, to drive two SFQ gates, a splitter [1] is required that reproduces the input pulse. Figure 3(a) shows the schematic of a splitter. Once the splitter receives a pulse on its *input A*, it will generate two SFQ pulses on both its outputs (*output B* and *C*).

In SFQ logic, a merger gate [1] makes two SFQ pulses drive the same pin possible. Figure 3(b) shows the schematic of a merger. When two pulses *A* and *B* arrive too close in time, there will be only one pulse on the *output C*. In this case, the early one will trigger a single pulse to be outputted at *C*, and the later one is dissipated through J_3 (*A*) or J_4 (*B*).

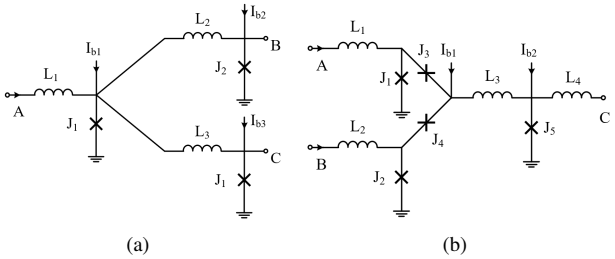

Figure 3. Schematic of (a) Splitter and (b) Merger Cell

G. Context for a SFQ Microprocessor

In this section, we provide a brief context for the SFQ-based CPU targeted in this paper. Given that SFQ is at a nascent stage in terms of design tools and manufacturing, our goal in this paper is to explore the design challenges of building a register file within the context of an in-order pipelined CPU, with fetch-decode-execute stages, based on RISC-V ISA with SFQ-based cells used in its design. We acknowledge that CMOS processor designs have much more complex control structures and execution paradigms (such as out-of-order and speculative execution). But a nascent technology requires research explorations such as those presented here to understand challenges and opportunities within a reasonable scale.

Throughout this paper, we rely on extensive physics and device-level characterization data for SFQ designs that our team has made publicly available on GitHub [16] as part of our IARPA superconducting design tools initiative. For example, the repository provides splitter and merger cell parameters, including the cell timings, power, and margin estimates, which are incorporated into our register file access pipeline models. In terms of architectural level evaluations, we provide a detailed Verilog-based implementation of HiPerRF and its peripheral access circuits, including the decoders and read/write port designs and wiring needs. For wiring, as we describe in more detail later, HiPerRF is primarily reliant on Passive microstrip Transmission Lines (PTL) and just a few Josephson Transmission Lines (JTLs) [17] for its design. Our place and route simulations account for these wire delays as well as any additional JJ counts needed. We then integrate our models into an in-order processor model to quantify the chip level are benefits of HiPerRF. Given the research resource limitation in our setting, while the fabrication of a chip is outside our scope, we believe that our detailed architectural exploration studies will provide key insights into the CPU level implications of future technologies such as SFQ.

III. Clock-Less NDRO Register File

This section describes our NDRO based register file design, which acts as a strong baseline. We first discuss how we design our innovative clock-less read, write and reset ports of the NDRO baseline register file using SFQ logic gates. The goal here is to demonstrate how to eliminate the need for clock distribution in SFQ register file design setting aside the memory density improvement of HC-DRO cells. In the next section, we show how the clock-less NDRO register file is enhanced with the dense HC-DRO cells to create the HiPerRF design.

Figure 4 shows the design. We show the design with one read and write port. Each rectangular box with an IN and OUT is one register entry. The figure shows a set of such register entries that together form the register file. At the output end, an explicit merger gate (marked as M) merges the output. Since there is a single read port, a single register is read enabled, which produces an output while all other registers do not place any pulses on the output. The output from the single register whose read is enabled will then be sent as R_DATA. The figure shows three blocks: read and write ports that are similar to CMOS design and an SFQ-specific reset port.

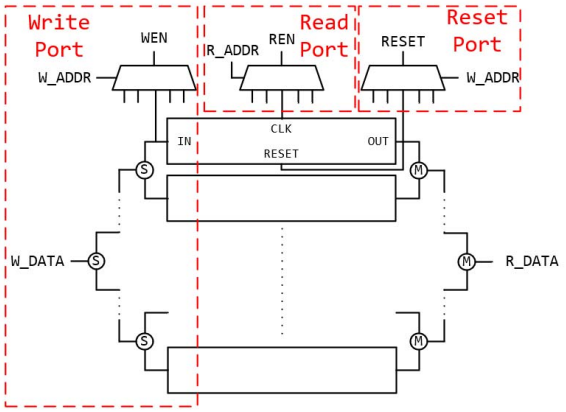

Figure 4. NDRO register file design

A. Read Port Design

A register read request is transformed into a read enable signal for the corresponding register. This transformation may be done using a demultiplexer (DEMUX) to decode the read address. CMOS designs may use combinational gates to achieve DEMUX functionality, as shown in Figure 6(a). However, in SFQ designs implementing such combinational design is prohibitively expensive.

Figure 5 shows the schematic of an AND gate for combinational design. It costs 12 JJs. To build the DEMUX, we have to split the input signal (IN) and the select signal (SEL) across the two AND gates. The NOT gate also costs 10 JJs and also needs a clock signal. As such, in SFQ technology the size of logic gates, the additional clock signals, and the existence of mergers and splitters will make the combinational DEMUX design very large. A 1-to-2 combinational DEMUX needs a total of about 50 JJs.

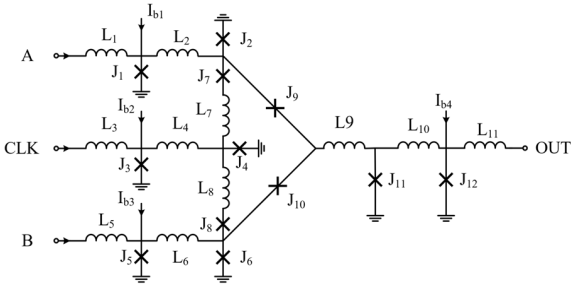

Figure 5. Schematic of an AND Gate

Rather than a combinational design, we built a demultiplexer using a Non-Destructive ReadOut cell with Complementary output (NDROC) which was proposed in prior work [7], [18]. Figure 6(b) shows the NDROC block diagram. The select signal (SEL) is connected to the SET input of the NDRO cell. If the SEL signal receives a clock pulse (a value 1), then the NDROC cell's SET pin is activated. When a pulse to the clock pin (CLK) is provided, it then outputs that pulse on the Q0 output (OUT0), and the complement is sent to Q1 output. Thus the NDROC can be repurposed to act as a 1-to-2 DEMUX. Note that in the above description, the SEL signal is driven by the source register number. The source register could be an architected register encoded in the instruction if no renaming is done. Otherwise, the source register is the physical register number.

The NDROC based 1-to-2 DEMUX costs only 33 JJs [19], which is about 60% of the combinational design based on AND gates. However, for this design to work correctly, the RESET signal needs to be asserted after each demux operation. Recall that the NDROC cell preserves any prior "1" stored in it from a select signal that deposited a "1". We need to clean the "1" if we want to write a "0" for the next selection.

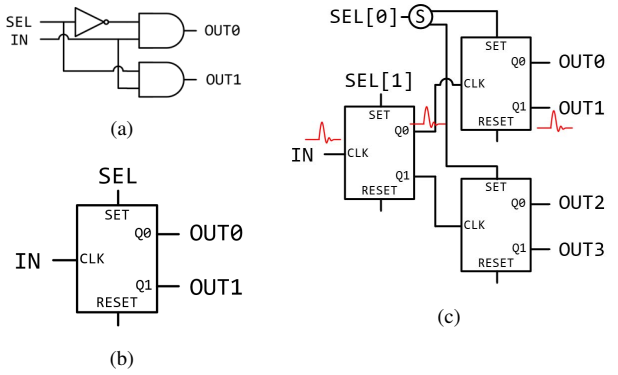

Figure 6. (a) DEMUX built with combinational logic (b) DEMUX built with NDROC (c) 1-to-4 DEMUX with NDROC

Since most practical ISA designs have many registers, it is necessary to build a 1-to-n DEMUX using NDROCs. The proposed hierarchical tree structure DEMUX is shown in Figure 6(c). The SEL[1] signal is connected to the first NDROC cell to either activate the top bank or bottom bank of the register file. The SEL[0] signal selects one of the two registers in either the top or the bottom bank in the second step. The SEL[0] signal must be split to drive the two NDROCs. The outputs of the first level NDROC are connected to the CLK pin of the second level NDROC.

To activate a register in a read operation, we connect the register number's select bits to the appropriate level of the NDROC tree. Then a single clock pulse is provided as the read enable pulse. The read enable pulse then traverses through the NDROC tree and will trigger the pulse on the OUT port corresponding to the register number.

To design a read port using the above design, we connect the read enable (REN) signal (generated from a decoder stage) to the starting input (IN) of the DEMUX and the read address (R_ADDR) (register number) to the SEL pins. Each of the output pins (OUT0..OUTN) is connected to the corresponding NDRO register entry. Since each NDRO register entry may have a 32/64-bit value, the corresponding OUT pin must be split 32/64 times in a tree hierarchy again to read the register's entire width. These splitters are omitted in Figure 4 for simplicity.

In this design, no clock distribution is needed since the read enable and read address signals provided as inputs act as triggers that move the fluxons through various gates to trigger the appropriate register to be read eventually. Using this design, we eliminate the need for clock distribution in the read port design. In fact, for the entire register file design, we use the enable signal as the trigger without the need for an explicit clock, thereby making our design scalable and robust to clock skew.

Finally, the output port merger block connects the output of each NDRO register to the output port of the register file.

B. Reset Port Design

In a CMOS design, it is possible to overwrite an existing memory cell content with a new value. However, in an SFQ memory cell that has an existing SFQ pulse (equivalent to storing a "1" in a cell), it is not possible to replace it with a "0". Writing a no pulse does not remove the existing SFQ pulse in the cell. Hence, every write operation must first reset all the bits in a register entry to "0" before a new write operation can be performed.

Every write operation to a NDRO register must be preceded by a reset so that the existing pulses in each memory cell are dissipated. Thus a new reset port is necessary for performing this function. Since the reset port is used prior to writing the data, the reset port uses the destination register's address (W_ADDR) to access the register. It also uses a special RESET_ENABLE signal that is sent as input to the DEMUX of the reset port. The DEMUX circuit moves the RESET_ENABLE pulse through the gates in the port to eventually select the destination register. The reset pulse reaches the selected register's reset pins so that the content of the selected register is set to "0."

C. Write Port Design

Once the reset operation is completed, the write port is activated to perform the write operation. The write port has three inputs, write enable (WEN), write address (W_ADDR), and write data (W_DATA). Same as the read and reset ports, the W_ADDR is decoded using the DEMUX design that we described. The write data is split and sent to all the NDRO registers. In order to write the data to the

correct NDRO registers, we add Dynamic AND (DAND) gates [13] between the write data and the SET pin of the NDRO register as shown in Figure 7(a).

DAND gates do not use clock signals to control the timing; instead, they use the hold time to control. Figure 7(b) shows the timing of the DAND gate. If two inputs arrive between the hold time, there will be a pulse on the output. Otherwise, these inputs will not generate any output. By using DAND gates, we can avoid using clocked AND gates to reduce the complexity of the design. When both WEN signals and W_DATA arrive within the hold time window, the pulse will arrive at the corresponding NDRO register's SET pin and write the data into the NDRO register. Thus the benefit of using DAND in the write port is to eliminate the need for clocking and splitter cells that may be needed for clock distribution.

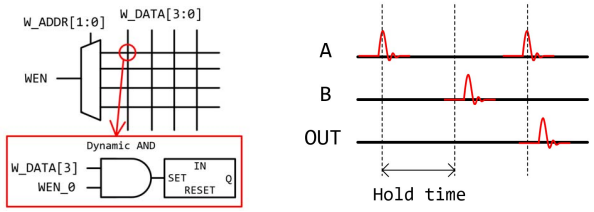

Figure 7. (a) NDRO write circuit (b) Dynamic AND timing

D. Read and Write Operation

Read operation: To initiate a read operation on the register file, the decoder extracts R_ADDR and sends that address along with REN (read enable) pulse to the register file. The REN pulse traverses the gates selected by the R_ADDR bits to move the register data out to the execution.

Write operation: First, the write operation must reset the NDRO register entry. The write operation sends the WEN and W_DATA to write the corresponding NDRO registers. A reset of all the NDROCs follows the write operation before the next operation can begin.

E. Timing

The register read and write operation control signals (REN, WEN, and RESET) are generated in the decode stage of the pipeline. While these three signals originate at the same time, they must be delivered with appropriate delays. Based on our detailed device modeling simulations (more details on modeling to follow), the NDROC gate in the current SFQ technology can receive two successive enable signals on its input (IN) with a 53ps delay. That means two read enable, write enable or reset signals must be at least 53ps apart. This delay is the cumulative delay of $Hold_{RESET}$ + $Critical_{RESET\_to\_SET}$ + $Setup_{SET}$ delay of the gates in the register file ports. The propagation delay, which is the time it takes for an SFQ pulse on the IN to reach the OUT, is about 24ps, and it is much less than 53ps. Hence the NDROC tree DEMUX can be fully pipelined at a cycle time of 53ps.

For any write operation, the reset signal has to precede any write operation. Our device-level simulation measured that critical time [20] between *RESET* signal and when data can be sent on the input *IN* of a register. This delay to separate the WEN from the RESET is 10ps, less than the 53ps delay needed for the DEMUX access. Based on these considerations, the clock cycle of the NDRO register file is 53 ps, while different control signals, such as RESET followed by WEN signals within a clock, are delayed by 10ps.

The timing for an example instruction sequence is shown in Figure 8. During the execution of one instruction, there is at most one register write operation, and at most two source register read operations. The figure shows a sequence of instructions labeled Inst 0 ... Inst x+1. Let us assume that the write back operation from Inst 0 is going to overlap with the source read operation of Inst x. Thus the write from old instruction (write back), which is the R1 from Inst 0, and two read operations from current instruction, R1 and R3 from Inst x will contend for the register file access. Inst X also has a read-after-write dependency with Inst 0 here.

Given this scenario, our preferred design option initiates the write operation of R1 before the read operation so that we can ensure the internal forwarding. The write operation first initiates a RESET operation. After RESET, the WEN (write enable) signal pulse is provided with a delay, which is based on the critical time between *RESET* and *IN* of the R1 register. Then the instruction initiates the REN (read enable) pulse. The second read operation for source register R3 in the second cycle is initiated concurrently with the RESET and WEN operation of Inst 1.

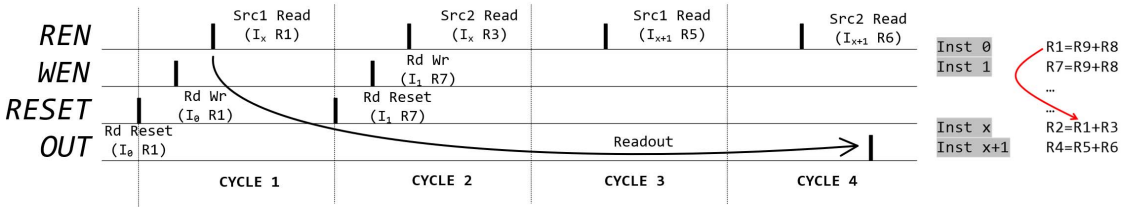

Figure 8. 32×32 bits NDRO register file timing

IV. HiPerRF: HC-DRO RF with NDRO Capabilities

HiPerRF uses HC-DRO cells to replace the NDRO cells to improve register file density. In our prior work we proposed HC-DRO cells [9] for density, but that work did not consider how to tackle the unwanted destructive readout property in designing a CPU register file. Our prior work also did not account for the circuit timing requirements and the various challenges associated with scheduling a register file's read/write operations.

Figure 9 shows the HiPerRF design. This design assumes that the HiPerRF design is a self-contained unit, and the rest of the CPU operates on each bit of information separately. Namely, even though the design stores 2-bits in one cell, they are read out as at most three pulses using an HC-READ circuit (described below) and fed to the rest of the logic. There are four major components in HiPerRF different from the baseline NDRO register file. First, the obvious replacement of NDRO cells with HC-DRO cells to store data. The second, the absence of a reset port. The third is a new output port design that enables the HiPerRF operation to retain the data. The last one is adding the HC-CLK, HC-WRITE, and HC-READ circuits that can decode and encode the two-bit storage HC-DRO cell into up to 3 separate pulses and vice-versa.

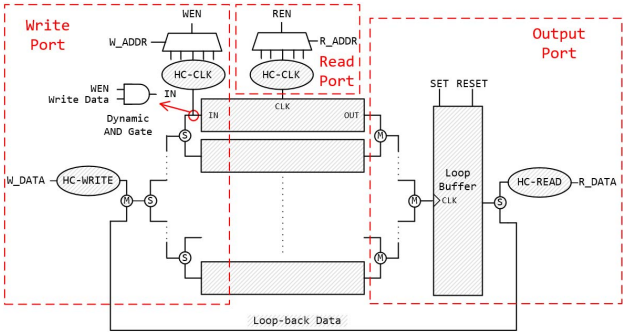

Figure 9. HiPerRF design

A. HC-DRO Read and Write Circuits

To read and write the HC-DRO cells correctly, we need to design the HC-DRO specific read and write circuits based on the initial design shown in previous work [9].

HC-WRITE circuits: Each HC-DRO cell encodes two bits of information into 0 to 3 pulses. We need an HC-WRITE circuit to convert the two bits of information generated by an ALU into up to 3 pulses for storage in HC-DRO. The HC-WRITE circuit designed is shown in Figure 10(a). The write circuit uses Josephson Transmission Lines (JTL), represented as diamonds with J in Figure 10(a). JTL is an SFQ design element that allows the fluxon to pass through it with delay. For instance, in the figure, when two pulses arrive at $B0$ (LSB) and $B1$ (MSB), the pulse starting at $B0$ goes through the two merge cells to produce the first pulse on the OUT. The pulse from B1 travels through three JTLs horizontally and goes through the splitter and merge cell to generate the second pulse. The split pulse from B1 will go through the vertical JTL path and eventually become the third pulse. The JTLs act as delay elements to create the minimum required separation to store two consecutive pulses into the HC-DRO cell; in our current design, this delay is about 10ps due to the requirement of the setup and hold time [21] of HC-DRO cells.

HC-CLK circuits: To read HC-DRO cells, we need to send three consecutive pulses to the input pin to read out all the fluxons stored inside. For instance, the REN signal that eventually reaches an operand register from the DEMUX port must generate three pulses to read each HC-DRO cell. The HC-CLK circuit is used to duplicate one SFQ pulse into three pulses. The circuit is shown in Figure 10(b). Same as before, the design uses JTLs to create three pulses that meet the required timing restrictions without any explicit clock signals.

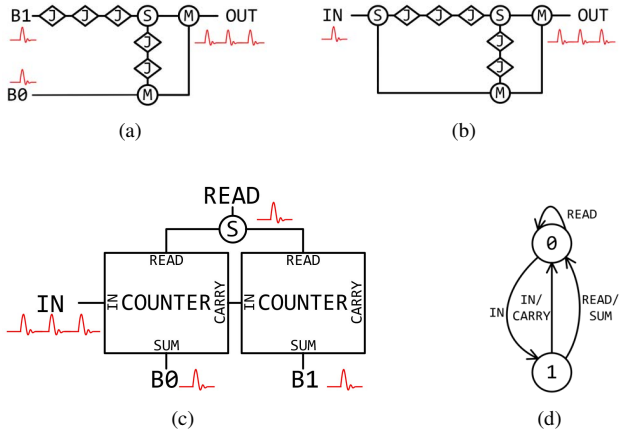

Figure 10. (a) HC-WRITE design (b) HC-CLK design (c) HC-READ design (d) state machine diagram of the counter

HC-READ circuits: Reading HC-DRO cells may produce 0 to 3 consecutive pulses at the input pin. These serial pulses need to be translated into normal one-bit logic to create two parallel pulses. The proposed HC-ReAD design used in this work is built using two one-bit counters [22] to build a two-bit counter. The design is shown in Figure 10(c). The state machine diagram of the counter is shown in Figure 10(d). After counting the pulses, the circuit generates two bit output as two parallel pulses on B1 and B0.

B. LoopBuffer for Non-destructive readout

The output port design of HiPerRF provides the non-destructive readout capabilities for HC-DRO cells. We add a LoopBuffer to the output port, a set of NDRO cells that enables restoring a register entry data after a read operation.

Read operation flow: The CLK input of the NDRO cell is connected to the output pulses produced from an HC-

DRO register. When an instruction wants to read a source register, the LoopBuffer's NDRO cell is first set to 1; namely, a single pulse is stored in the NDRO cell prior to the start of the register read operation. Each of the 2-bit source register values (at most three pulses stored in a single HC-DRO cell) arrive at the CLK pin of the LoopBuffer's NDRO cell. The incoming pulses from the HC-DRO register exit the LoopBuffer as output pulses. For instance, if the HC-DRO cell has an encoded value of "10", it will generate two output pulses, triggering the LoopBuffer NDRO to produce two output pulses. If the HC-DRO cell has an encoded value of "11", it will generate three output pulses, triggering the NDRO to produce three output pulses.

Those output pulses go through the splitter, and one branch of that restores the data back to the source register, while the other branch is sent to HC-READ to decode into two-bit values for operation by the ALU.

Write operation: In HiPerRF, when an instruction enters the write back stage, its destination register write operation is divided into two steps: the first operation reads the register content and erases that register content using the LoopBuffer. To enable this erase operation, the LoopBuffer is first reset to zero and the current content of the destination register is read into the LoopBuffer to dissipate its value. Then the write back of the new value follows normally since the destination register is now cleared.

The LoopBuffer design is based on the intuition that only a few registers are actively read at any given time window. Hence, one needs to preserve the non-destructive read property only for those active registers. As such, our design allows a large HC-DRO register file to share a small victim NDRO buffer that helps the readout data to recycle back to the original register.

C. Read Port and Write Port Design

The read and write port operate similarly to the NDRO baseline design. The one difference is adding the HC-CLK circuits between the DEMUXs and HC-DRO cells. The HC-CLK generates three pulses for read enable and write enable signals. This operation enables us to read each HC-DRO cell using the enable pulses generated from HC-CLK. The use of NDRO cells in LoopBuffer provides interesting optimization opportunities. As explained above, the NDRO cell can be reset to erase register content. We use this property to use a single read port to work as a reset port as well. Thus the need for a reset port is eliminated in the HiPerRF design.

Unlike the NDRO register file's write port, the HiPerRF's write port needs to accept data both from the regular register write operations and the LoopBuffer. Hence, a new merger gate is added at the write port, as shown in Figure 9. An HC-WRITE is added between the input and the merger to encode the data for HC-DRO storage.

D. Timing

Similar to the NDRO RF, the bottleneck in HiPerRF is also the NDROC of the DEMUX. The gap between two REN signals and two WEN signals is also 53ps, which will be the cycle time. The control pulse timing of HiPerRF is shown in Figure 11. At the start of executing Instruction X, a write operation of the destination register is initiated. The write operation first generates a REN pulse (to reset the register). The REN pulse passes through HC-CLK to generate three pulses (each 10ps apart in our design due to the requirement of the setup and hold time of HC-DRO cells, as shown in three rectangular pulses in the figure). The WEN pulse follows this reset operation in the second clock, which passes through HC-CLK to generate three pulses. Concurrent with the WEN pulse, the first source read operation is initiated with a REN pulse in the second clock (for clarity, the three pulse sequence is shown as just one pulse). The loopback write for this read operation is initiated in the third cycle, as the dashed arrow shows. In the third clock, the second source register's REN pulse is initiated, followed by a loopback write operation of source 2 in the fourth clock. During the fourth clock, the second instruction's destination register's write operation is also initiated, and the process repeats every three cycles. Note that in this timing sequence, the write operation of register R1 is unable to forward the data to Inst X. Hence, Inst X has to go through the full source register read operation.

The loopback write brings one more issue to the forefront: the Read-After-Read (RAR) hazards. If Inst x is reading from the R3 twice (R2=R3+R3), the second read operation at cycle three will read out nothing since the data has not come back yet. In this case, the second R3 should be duplicated from the first read operation rather than being read from the RF. Note that for precise exception handling, the loopback write operation cannot be optimized even if the same register is being overwritten anyway.

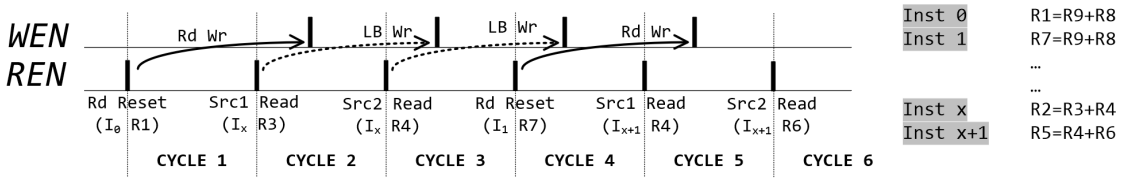

Figure 11. Timing of HiPerRF

V. Multi-Bank HiPerRF

For performance reasons, we considered adding an additional read port to HiPerRF. However, in the HiPerRF design adding a read port requires performing the two concurrent loopback operations as well, which require two write ports. Hence the HiPerRF design essentially requires read and write ports to scale together. Based on our design estimate, a 32x32 bits HiPerRF with two read ports and two write ports costs nearly triples the JJ counts due to superlinear increase in the merger, splitter, and other peripheral circuitry needed to support two ports. An area-efficient solution is a banked register file. By splitting HiPerRF into two banks, we can achieve two read ports and two write ports without the superlinear growth in the peripheral circuitry. The banked design also reduces the DEMUX depth, speeding up access to the register file. Since each bank has only half the registers, the banked design removes one merger and one splitter (about 10ps time) from the loopback path, saving loopback timing. Thus banking not only reduces port contention but also reduces the loopback path delay.

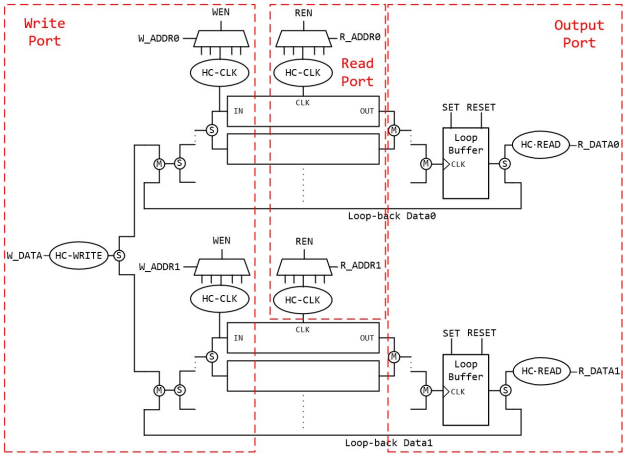

Figure 13. Dual-banked HiPerRF design

A. Port Design

The dual-banked HiPerRF design is shown in Figure 13. Each bank will have its own read port, write port, and output port. Although the number of the DEMUXes is doubled, each of the DEMUXes is only half the size of the one in the HiPerRF design above. Hence the main JJ overhead is the extra LoopBuffer. We omit the splitters for illustration simplicity. The rest of the circuits remains the same as the HiPerRF design mentioned in section IV.

B. Timing

Similar to the HiPerRF design, the main bottleneck is still the DEMUX. The cycle time remains 53ps here. The timing is shown in Figure 12. However, since we may perform two read operations in the same cycle, the scheduling will be different here. We split the register file into two banks based on the parity of the register number. Registers with odd register numbers belong to bank 0, and the other belongs to bank 1.

At the beginning of cycle 2, since Inst x needs to read the registers from different banks, we can send two read signals to both banks. Cycle one and cycle three are reserved for the write back's reset operation (Inst 1 and 3). Similarly, we send the read signal of R4 at the beginning of cycle 4. However, unlike HiPerRF, we do not send the next read signal in the next cycle. Instead, we reserve this for the write back's reset operation of Inst 2. The second read operation starts at the beginning of cycle 6. As a result, if the instruction is reading two registers from different banks, it only takes two cycles. However, if the instruction is reading two registers from the same bank, it takes four cycles.

In comparison, HiPerRF needs three cycles for all instructions. By doing so, we utilize the two read ports without increasing the complexity of the scheduling unit. Similar to HiPerRF, reading two same registers (R2=R3+R3) needs the duplication of the readout results.

VI. Evaluation

The evaluation of HiPerRF focuses on two aspects, hardware design evaluation and software simulation results for measuring application-level performance. For each of the above-described register file designs, we built Verilog netlists using the publicly available cell libraries [16]. We successfully verified the functionality and timing with different combinations of inputs, including accounting for wire

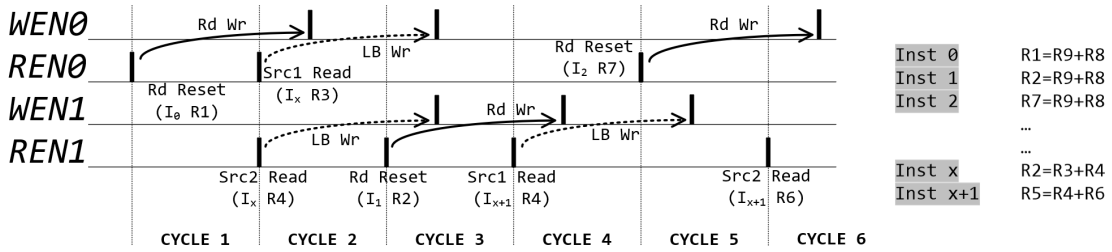

Figure 12. Timing of dual-banked HiPerRF

delays post place and route. We then integrated the Verilog model of our register file designs within the RISC-V Sodor core CPU to measure the overall JJ count reduction for the whole chip.

A. Hardware performance

The hardware performance is evaluated in two parts, JJ count and static power. The total JJ count is calculated by using the SFQ cell library provided by [16]. For the dynamic AND gate and NDROC, the data is derived from [13] and [19]. The static power for the entire register file design is also derived from the cell libraries. Note that in SFQ designs, the limiting design factor is the number of JJs that can be integrated with current fabrication technologies. Hence, measuring savings in terms of JJ counts is more appropriate than chip area. Nonetheless, the register file size is about 20% of the total CPU design area using NDRO cells, and it is reduced with HiPerRF by various amounts based on the size of the register file.

	Total JJ Count			**Percentage of Baseline**		
Register File Size (bits)	4 × 4	16 × 16	32 × 32	4 × 4	16 × 16	32 × 32
NDRO RF (Baseline Design)	784	9850	36722	100%	100%	100%
HiPerRF	695	5195	16133	88.65%	52.74%	43.93%
Dual-banked HiPerRF	736	5626	17094	93.88%	57.12%	46.55%

Table I
TOTAL JJ COUNT AND THE PERCENTAGE OVER THE BASELINE DESIGN

	Static Power (μW)			**Percentage of Baseline**		
Register File Size (bits)	4 × 4	16 × 16	32 × 32	4 × 4	16 × 16	32 × 32
NDRO RF (Baseline Design)	170.73	1997.49	7262.17	100%	100%	100%
HiPerRF	149.16	1220.05	3911.00	87.37%	61.08%	53.85%
Dual-banked HiPerRF	148.47	1289.89	4077.88	87.00%	64.58%	56.15%

Table II
STATIC POWER AND THE PERCENTAGE OVER THE BASELINE DESIGN

Table I shows the JJ count for each of the described register file designs for two different register file sizes. The data includes the JJ counts for splitters, mergers, and any necessary JTLs for the register file access. The first row shows the baseline, an NDRO based register file. The second row shows HiPerRF design. The third row shows a dual-banked HiPerRF design. The 4×4, 16×16 and 32×32 bits HiPerRF saves about 11%, 47%, and 56% of JJs compared to the baseline design. The extra JJs required to implement HC read and write circuits can be amortized with the density advantage of HC-DRO cells. Hence, the relative advantage of HiPerRF grows as the size of the register file increases in the future.

	Readout Delay (ps)			**Percentage of Baseline**		
Register File Size (bits)	4 × 4	16 × 16	32 × 32	4 × 4	16 × 16	32 × 32
NDRO RF (Baseline Design)	77	144	177.5	100%	100%	100%
HiPerRF	122.8	187.8	220.3	159.48%	130.42%	124.11%
Dual-banked HiPerRF	94.8	159.8	192.3	123.12%	110.97%	108.33%

Table III
READOUT DELAY AND THE PERCENTAGE OVER THE BASELINE DESIGN

Table II shows the static power for each design and the percentage of the baseline design. As expected, the static power is a function of the number of JJs used in a design. Hence, the HiPerRF with 32×32 bits consumes about 46% less static power compared to the baseline design, reflecting the reduction in JJ count. Note that these results did not include the benefits associated with reduced cooling power due to reduced static power. Heat extraction is a major power source and may lead to two orders of magnitude more energy consumption [8].

Full Chip Benefit: To quantify the benefits at the chip level, we synthesized the RISC-V Sodor in-order core with HiPerRF by using the qPalace tool [23] to get the JJ count. The Sodor core has five main parts: ALU, Register File (RF), Control and Status Registers (CSR), control path, and front end. The total JJ count of these various CPU components using baseline NDRO register file design is 139,801. When the register file is replaced with HiPerRF, including all the additional overheads of read/write and clock circuits, the total JJ count reduces to 117,039 JJs. Thus the total JJ reduction is 16.3%.

We also analyzed the readout delay, which is a critical performance metric. Table III shows the readout delay for each design and the percentage of the baseline design. The 4×4, 16×16 and 32×32 bits HiPerRF actually increases the readout delay due to the need to write the data into the LoopBuffer before the data is made available to the ALU. The increase in delay is about 24% for a larger register file. However, the dual-banked design could reduce the readout delay overhead to 8% by reducing the long latency of accessing the NDROC. We expect that as the register file size grows, relative gains in power and JJ count reduction of HiPerRF grow. That is because the overhead circuitry costs are better amortized. Moreover, even the readout delay overhead will eventually match the baseline with a larger size.

B. Simulation Results

To analyze the performance improvement at the application level, we designed an SFQ based gate-level CPU simulator. The ISA we chose to simulate is RISC-V 32I, which is the basic RISC-V ISA. The simulator is based on the RISC-V ISA Simulator Spike [24] and written in C++. While the basic simulator uses a function-level pipeline, our enhanced simulator implements a gate-level pipeline. Such a gate-level pipeline model is necessary, for instance, to enable pipelined execution within the read and write ports that use a long chain of NDROC cells. Our simulator

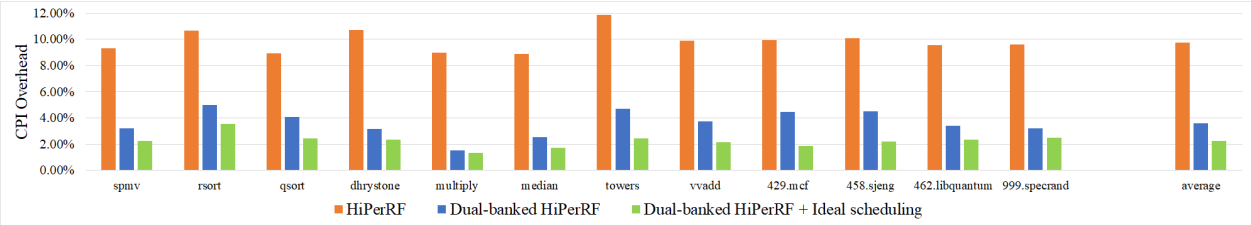

Figure 14. CPI overhead over baseline (NDRO RF) of different RISC-V and SPEC 2006 benchmarks for different designs

uses the notion of a macro clock to simulate the fetch-decode-execute-writeback pipelines, and it uses micro clocks to simulate the gate-level pipelining functions of SFQ. As we discussed before, our simulator supports the internal forwarding within the register file when appropriate without violating timing. In our current simulator design, an external cache at 77K is interfaced with the SFQ design, which is the usual practice for interfacing larger memories [25]. As such, all memory references are satisfied from the 77K memory. However, recent advances in new materials such as three-terminal JJs, magnetic JJs, ferromagnetic JJs, and spin-based JJs have advanced memory integration closer to SFQ [26]–[29], which is outside the scope of our simulation capability.

To get the depth of each gate-level pipelined stage, we synthesized the RISC-V Sodor in-order core Verilog code by using the qPalace tool [23] which synthesizes SFQ designs built from the cell libraries. We measured the gate-level pipeline depth of each functional block from the synthesis results. Based on the synthesis result from qPalace, the Sodor design has a worst-case gate-level cycle time of 28ps, which is about half of the 53ps needed for HiPerRF. Hence, we used a cycle time of 28ps for each gate, and each read or write operation takes two cycles. As such, the readout delay shown in Table III is translated into the corresponding number of cycles. These readout delays are then used as input to determine the stall cycles for handling dependencies. The performance results below account for any write port contention between loopback writes, and the traditional write back path. The way it is accounted for is described in Section IV-D where we only issue instructions every three cycles, where one of those cycles is reserved for write back operation to not conflict with loopback. Similarly, we also account for the latency in the dual-banked HiPerRF described in V-B. Hence, by design, our approach statically eliminates write port contention between write back and loopback, and the scheduling costs are modeled in the simulator accurately.

The benchmarks we used are from the RISC-V repository [30] and SPEC CPU 2006 [31]. Due to the ISA and simulator limitations, we ran 429.mcf, 458.sjeng, 462.libquantum, and 999.specrand. Some limitations include lack of full support for floating-point instructions, cross-compiler failures from the GCC compiler. Due to the extremely slow gate-level simulations, each benchmark is simulated for 24 hours. SFQ simulators need to do detailed gate-level pipeline simulations to bring the SFQ pulse state properly loaded into the simulator. Hence, fast-forwarding using pinpoints [32] and other advanced simulation strategies need a careful redesign, which is part of our future work.

Figure 14 shows the CPI overhead for each benchmark and the average CPI overhead across all benchmarks, compared to the NDRO register file baseline. It is worth noting that in SFQ based designs, the gate-level pipelining poses significant challenges for read-after-write (RAW) hazards. The execution stage of the RISC-V core is 28 stages deep. Hence, any two instructions with RAW dependency in a short window will stall in the execution stage. We believe that current compiler optimizations may place RAW dependencies somewhat closer to each other to exploit data forwarding capabilities in traditional CPU pipelines. However, SFQ based CPUs require quite the opposite – to spread the RAW dependency instructions as far apart as possible. As a result, the average cycles per instruction measured in our modified RISC-V core gate-level simulator is about 30 cycles averaged across all the benchmarks. Furthermore, the compiler we used is not optimized for accessing multiple banks. Hence, for the dual-banked design, we also run the simulations that consider the ideal situation, in which all instructions read the two source registers from different banks.

The CPI of HiPerRF is about 9.8% worse than the baseline. As discussed earlier, HiPerRF's design goals are to use fewer JJs without significantly impacting performance. HiPerRF is expected to have somewhat lower cycle level performance than the baseline design because the LoopBuffer is in the critical path, and any subsequent instruction that wants to read the same source register may have to wait for the loopback write before accessing the register file. Dual-banked HiPerRF reduces this overhead to 3.6%. This improvement is due to less port contention and short readout delay. In the ideal situation, the CPI overhead is only 2.3%, which is almost as good as the baseline design but saves 53% JJs in the register file, and 16.3% fewer JJs even at the overall chip level.

C. Impact of Wire Delay

There are two types of wiring in RSFQ technology, Passive microstrip Transmission Lines (PTL) and Josephson Transmission Lines (JTL). Our design uses PTLs for all the wires, and we use JTLs only when there is a need to induce delays, as was the case with the HC-READ circuit. We have done the placement and routing of our design by using Cadence Innovus with the library extracted from the open-source qPalace tool [23]. Figure 15 shows the fully placed and routed results of HiPerRF. The white areas in Figure 15 show the LoopBack path of HiPerRF. The longest delay on the LoopBack path is only 4.6ps, which is much smaller than the decoder latencies (53ps). Although the LoopBack path looks long in the visual illustration in Figure 9, this path is quite short in reality after placement and routing are done. On average, across multiple circuits placed and routed with qPalace, the wire length between two gates is 262μm between any two gates. Furthermore, as per the qPalace derived data, the delay of the PTL is 1ps/100μm, so the average wire delay between two gates is 2.62ps. Based on these data, the new readout delay accounting for all wire delays is shown in Table IV. With a detailed wire delay inclusion, the overall readout latency overhead increases by about 1% compared to the baseline; hence the CPI performance impact is at most 1%.

	NDRO RF (Baseline Design)	HiPerRF	Dual-banked HiPerRF
Readout Delay (ps)	216.8	270.1	236.8
Loopback Latency (ps)	–	108.4	93.7

Table IV
READOUT DELAY, LOOP-BACK LATENCY WITH PTL DELAY

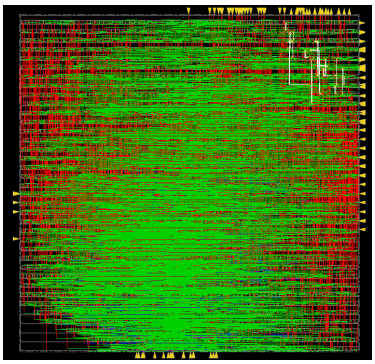

Figure 15. Placement and routing results of HiPerRF

VII. Related Work

While we have described the related work as related to specific design choices throughout the text, in this section, we focus on a couple of related works that focus on superconducting register file designs.

Fujiwara [11] proposed a shift register file built with DRO cells. Their design is a basic single-bit shift register design where the data moves from the tail to the head of the shift register. They used their design in the functional verification without considering the architectural and performance implications in a CPU. However, the implications of trying to build an HC-DRO register file design with loopback capabilities are significant. (1) HiPerRF is designed to function with the 2-bit DRO cell, which is not considered in prior work. As such HiPerRF design must also handle multi-bit decoding and encoding circuits, which are novel and entirely outside of the scope of [11]. (2) HiPerRF design also carefully considers the timing challenges of scheduling loop back and write back data arriving at the write port. (3) We also demonstrate how read ports can be repurposed as reset ports to remove data from an HC-DRO register before a new write is allowed. This design substantially simplifies the cost of the port design since we eliminate many of the splitter cells that are latency-sensitive. (4) We compared our design with the NDRO design. We evaluated our design and showed the actual JJ count benefits with a little performance penalty, which is negligible with the dual-bank option.

Dorojevets and Chen [33] proposed a Reciprocal Quantum Logic NDRO-based storage. They described the design details and extended their design to different memory designs, such as register file and cache. However, their design is fundamentally limited to using NDRO cells to store register data. Hence, they do not consider the possibility of reducing register file design size using HC-DRO cells as the primary storage elements.

VIII. Conclusion

Superconducting devices based on Josephson Junctions are an important device technology that needs to be explored in the context of a microprocessor design. The development of single flux quantum (SFQ) devices and the logic design tools and techniques around SFQ has been gaining traction with significant research investments. In this work, we explored how SFQ based designs can be used to implement an area-efficient register file in the context of a modified RISC-V in-order core processor. Since memory is a premium resource in SFQ designs, we propose HiPerRF, which uses High Capacity Destructive ReadOut (HC-DRO) cells and yet supports the multiple read property critical for any microprocessor register file. We designed the HiPerRF and proposed multiple enhancements to reduce the access latency of HiPerRF. Using gate-level synthesis tools and gate-level simulations, we demonstrate that HiPerRF saves nearly 56% of the JJ counts. We executed a range of benchmarks on a pipelined CPU design with gate-level detail to show that the HiPerRF only pays about 10% performance reduction due to the LoopBuffer. We also demonstrate how to use a dual-banked design to reduce the port contention and reduce performance penalties of accessing DEMUX trees that are

embedded in the read, write, and reset ports of SFQ register files.

ACKNOWLEDGMENT

The research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the U.S. Army Research Office grant W911NF-17-1-0120. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein.

REFERENCES

[1] 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*, vol. 1, no. 1, pp. 3–28, 1991.

[2] O. A. Mukhanov, "Energy-efficient single flux quantum technology," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. 760–769, 2011.

[3] W. Chen, A. Rylyakov, V. Patel, J. Lukens, and K. Likharev, "Rapid single flux quantum t-flip flop operating up to 770 ghz," *IEEE Transactions on Applied Superconductivity*, vol. 9, no. 2, pp. 3212–3215, 1999.

[4] M. Dorojevets, C. L. Ayala, N. Yoshikawa, and A. Fujimaki, "8-bit asynchronous sparse-tree superconductor rsfq arithmetic-logic unit with a rich set of operations," *IEEE transactions on applied superconductivity*, vol. 23, no. 3, pp. 1 700 104–1 700 104, 2012.

[5] T. Filippov, M. Dorojevets, A. Sahu, A. Kirichenko, C. Ayala, and O. Mukhanov, "8-bit asynchronous wave-pipelined rsfq arithmetic-logic unit," *IEEE transactions on applied superconductivity*, vol. 21, no. 3, pp. 847–851, 2011.

[6] K. Fujiwara, H. Hoshina, Y. Yamashiro, and N. Yoshikawa, "Design and component test of sfq shift register memories," *IEEE transactions on applied superconductivity*, vol. 13, no. 2, pp. 555–558, 2003.

[7] 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*, vol. 26, no. 5, pp. 1–5, 2016.

[8] I. C. o. S. Scott Holmes, "Superconductor electronics technology roadmap for irds 2020," *Applied Superconductivity Conference*, 2020.

[9] N. K. Katam, H. Zha, M. Pedram, and M. Annavaram, "Multi fluxon storage and its implications for microprocessor design," in *Journal of Physics: Conference Series*, vol. 1559, no. 1. IOP Publishing, 2020, p. 012004.

[10] K. I. Farkas, N. P. Jouppi, and P. Chow, "Register file design considerations in dynamically scheduled processors," in *Proceedings. Second International Symposium on High-Performance Computer Architecture*. IEEE, 1996, pp. 40–51.

[11] K. Fujiwara, Y. Yamashiro, N. Yoshikawa, Y. Hashimoto, S. Yorozu, H. Terai, and A. Fujimaki, "High-speed test of sfq-shift register files using ptl wiring," *Physica C: Superconductivity*, vol. 412, pp. 1586–1590, 2004.

[12] M. Tanaka, K. Takata, T. Kawaguchi, Y. Ando, N. Yoshikawa, R. Sato, A. Fujimaki, K. Takagi, and N. Takagi, "Development of bit-serial rsfq microprocessors integrated with shift-register-based random access memories," in *2015 15th International Superconductive Electronics Conference (ISEC)*. IEEE, 2015, pp. 1–3.

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

[14] N. K. Katam and M. Pedram, "Logic optimization, complex cell design, and retiming of single flux quantum circuits," *IEEE Transactions on Applied Superconductivity*, vol. 28, no. 7, pp. 1–9, 2018.

[15] J. A. Delport, K. Jackman, P. Le Roux, and C. J. Fourie, "Josim—superconductor spice simulator," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–5, 2019.

[16] L. Schindler, "Rsfq cell library." [Online]. Available: https://github.com/sunmagnetics/RSFQlib

[17] Y. Hashimoto, S. Yorozu, Y. Kameda, and V. K. Semenov, "A design approach to passive interconnects for single flux quantum logic circuits," *IEEE transactions on applied superconductivity*, vol. 13, no. 2, pp. 535–538, 2003.

[18] M. Tanaka, R. Sato, Y. Hatanaka, and A. Fujimaki, "High-density shift-register-based rapid single-flux-quantum memory system for bit-serial microprocessors," *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 5, pp. 1–5, 2016.

[19] Y. Yamanashi, S. Nakaishi, A. Sugiyama, N. Takeuchi, and N. Yoshikawa, "Design methodology of single-flux-quantum flip-flops composed of both 0-and π-shifted josephson junctions," *Superconductor Science and Technology*, vol. 31, no. 10, p. 105003, 2018.

[20] C. J. Fourie, "Extraction of dc-biased sfq circuit verilog models," *IEEE Transactions on Applied Superconductivity*, vol. 28, no. 6, pp. 1–11, 2018.

[21] B. Zhang and M. Pedram, "qsta: A static timing analysis tool for superconducting single-flux-quantum circuits," *IEEE Transactions on Applied Superconductivity*, vol. 30, no. 5, pp. 1–9, 2020.

[22] T. Onomi, T. Kondo, and K. Nakajima, "High-speed single flux-quantum up/down counter for neural computation using stochastic logic," in *Journal of Physics: Conference Series*, vol. 97, no. 1. IOP Publishing, 2008, p. 012187.

[23] C. J. Fourie, K. Jackman, M. M. Botha, S. Razmkhah, P. Febvre, C. L. Ayala, Q. Xu, N. Yoshikawa, E. Patrick, M. Law *et al.*, "Coldflux superconducting eda and tcad tools project: Overview and progress," *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–7, 2019.

[24] RISC-V, "riscv-software-src/riscv-isa-sim: Spike, a risc-v isa simulator." [Online]. Available: https://github.com/riscv/riscv-isa-sim

[25] F. Ware, L. Gopalakrishnan, E. Linstadt, S. A. McKee, T. Vogelsang, K. L. Wright, C. Hampel, and G. Bronner, "Do superconducting processors really need cryogenic memories? the case for cold dram," in *Proceedings of the International Symposium on Memory Systems*, 2017, pp. 183–188.

[26] S. Shafraniuk, I. Nevirkovets, and O. Mukhanov, "Modeling computer memory based on ferromagnetic/superconductor multilayers," *Phys. Rev. Applied*, vol. 11, p. 064018, Jun 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.11.064018

[27] V. V. Ryazanov, V. V. Bol'ginov, D. S. Sobanin, I. V. Vernik, S. K. Tolpygo, A. M. Kadin, and O. A. Mukhanov, "Magnetic josephson junction technology for digital and memory applications," *Physics Procedia*, vol. 36, pp. 35–41, 2012.

[28] N. O. Birge, A. E. Madden, and O. Naaman, "Ferromagnetic josephson junctions for cryogenic memory," in *Spintronics XI*, vol. 10732. International Society for Optics and Photonics, 2018, p. 107321M.

[29] N. O. Birge and M. Houzet, "Spin-singlet and spin-triplet josephson junctions for cryogenic memory," *IEEE Magnetics Letters*, vol. 10, pp. 1–5, 2019.

[30] RISC-V, "riscv-software-src/riscv-tests." [Online]. Available: https://github.com/riscv/riscv-tests

[31] SPEC, "Spec cpu 2006." [Online]. Available: https://www.spec.org/cpu2006/

[32] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi, "Pinpointing representative portions of large intel® itanium® programs with dynamic instrumentation," in *37th International Symposium on Microarchitecture (MICRO-37'04)*. IEEE, 2004, pp. 81–92.

[33] M. Dorojevets and Z. Chen, "Fast pipelined storage for high-performance energy-efficient computing with superconductor technology," in *2015 12th International Conference & Expo on Emerging Technologies for a Smarter World (CEWIT)*. IEEE, 2015, pp. 1–6.