



BUILDING SFQ CPU: ONE MICROARCHITECTURE BLOCK AT A TIME

Murali Annavaram

USC

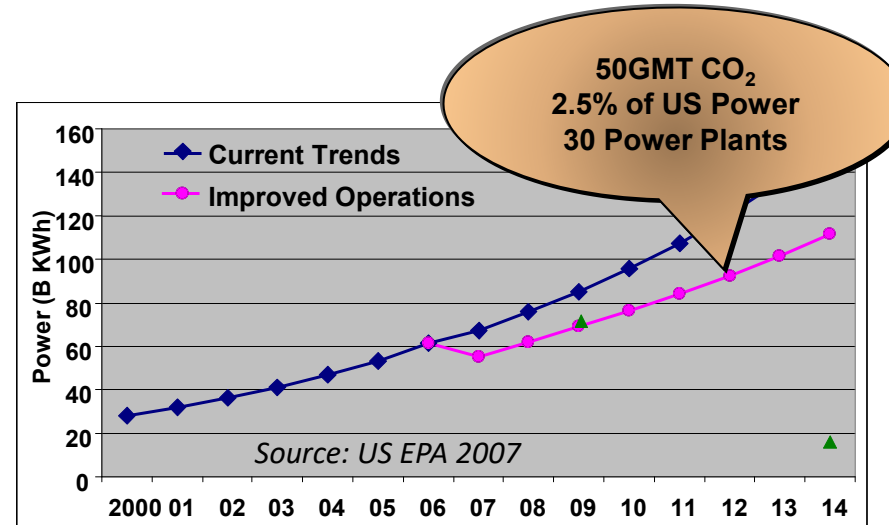
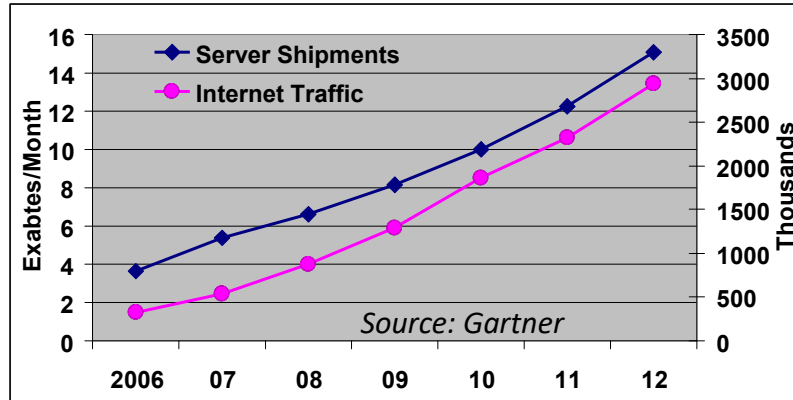
<https://Annavar.am>

K-ARCH, MICRO

Oct 18th 2025

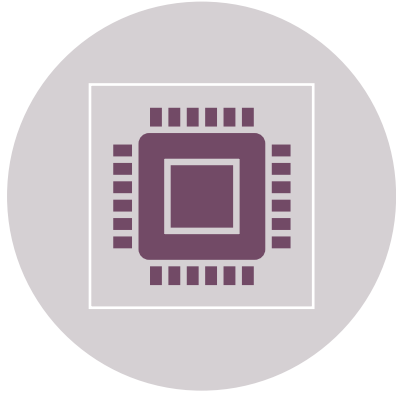
<https://discoverexpedition.usc.edu/>

CMOS Technology Trends



- Unprecedented societal transformations enabled by ICT
- Computing sits at the heart of this transformation and must keep pace to enable continued growth
- Current energy and environmental cost trend of CMOS based designs are unsustainable
- Need a paradigm shift in computing to put us on a more sustainable/scalable ICT

CMOS Challenges



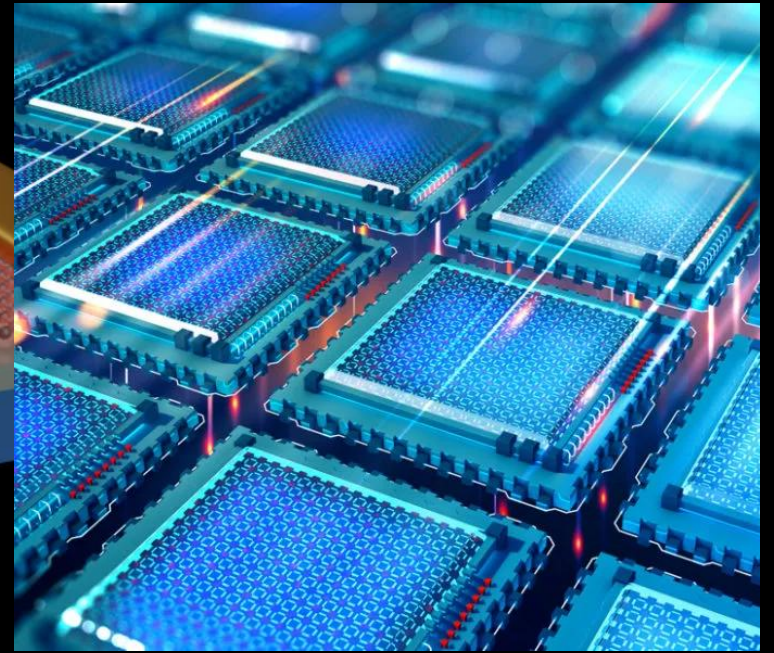
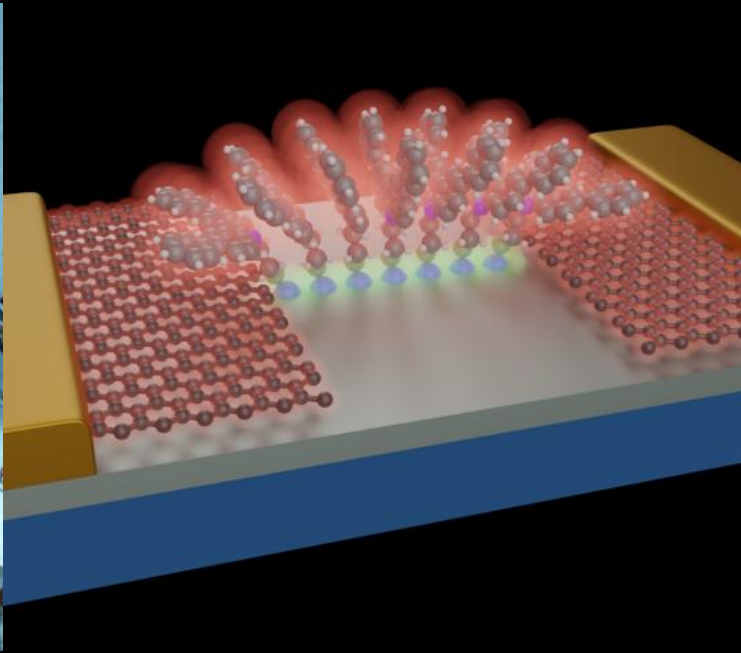
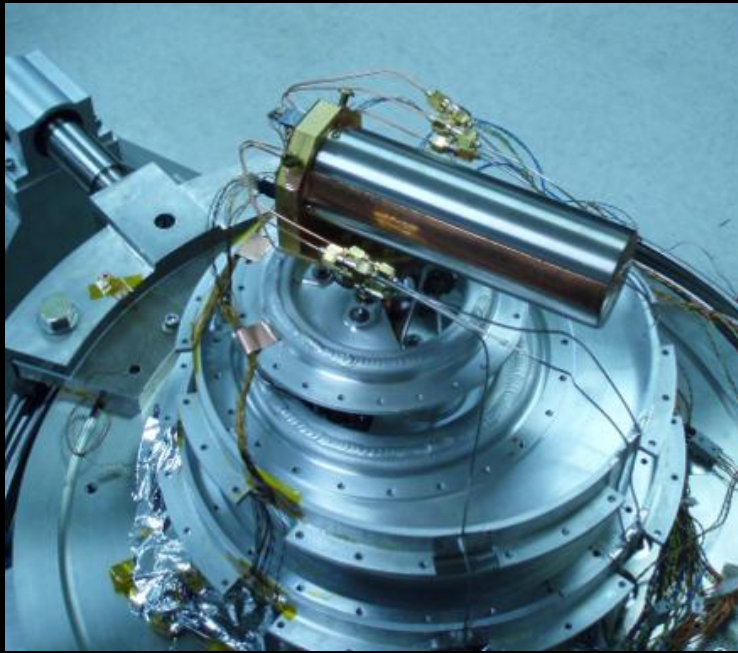
CMOS TECHNOLOGY HAS
BEEN DEVELOPED FOR
DECADES



DENNARD SCALING AND
PHYSICAL SCALING
CHALLENGES ARE TOO LARGE

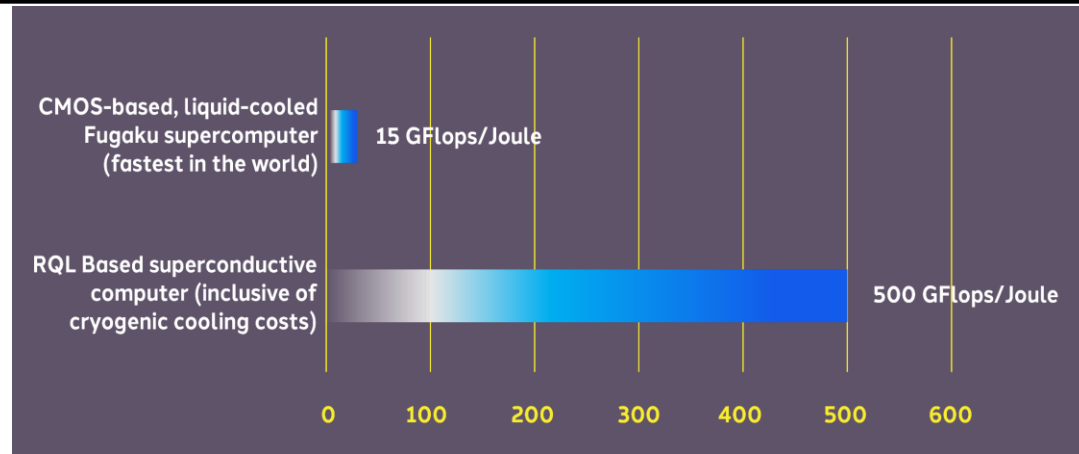


DENSITY WITHOUT DENNARD
== POWER & HEAT STRUGGLES

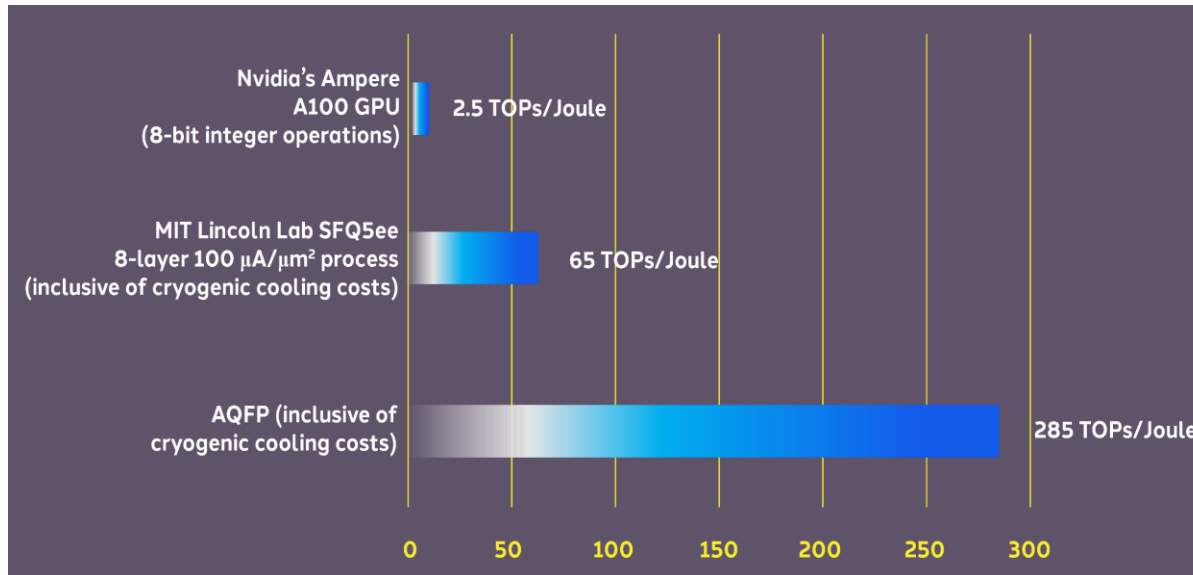


BEYOND CMOS

The Potential for Superconductor Circuits



System-level energy consumption (GFlops/Joule)



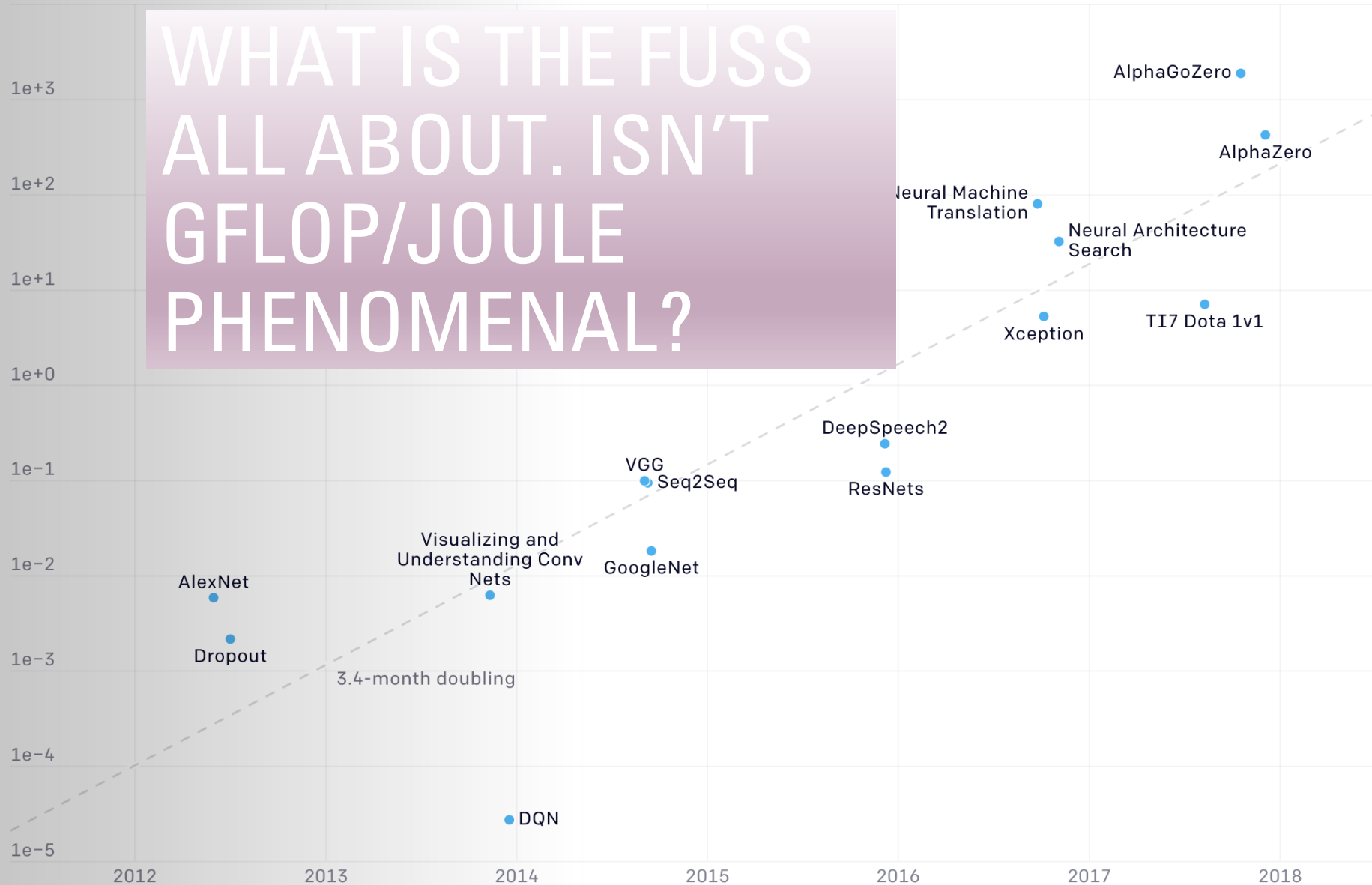
Neural network inference energy consumption (TOPs/Joule)

5

AlexNet to AlphaGo Zero: A 300,000x Increase in Compute (Log Scale)

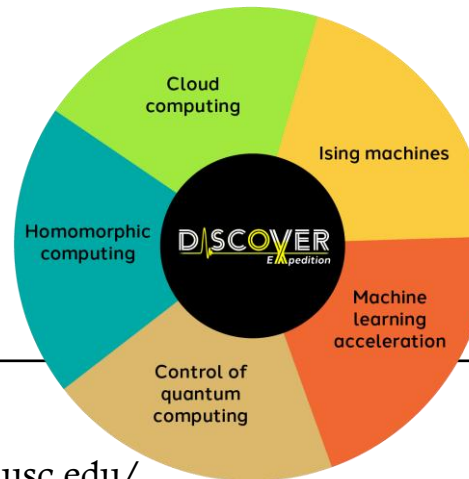
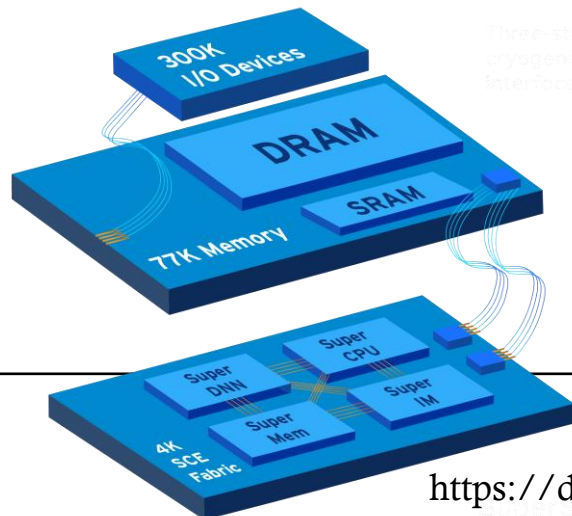
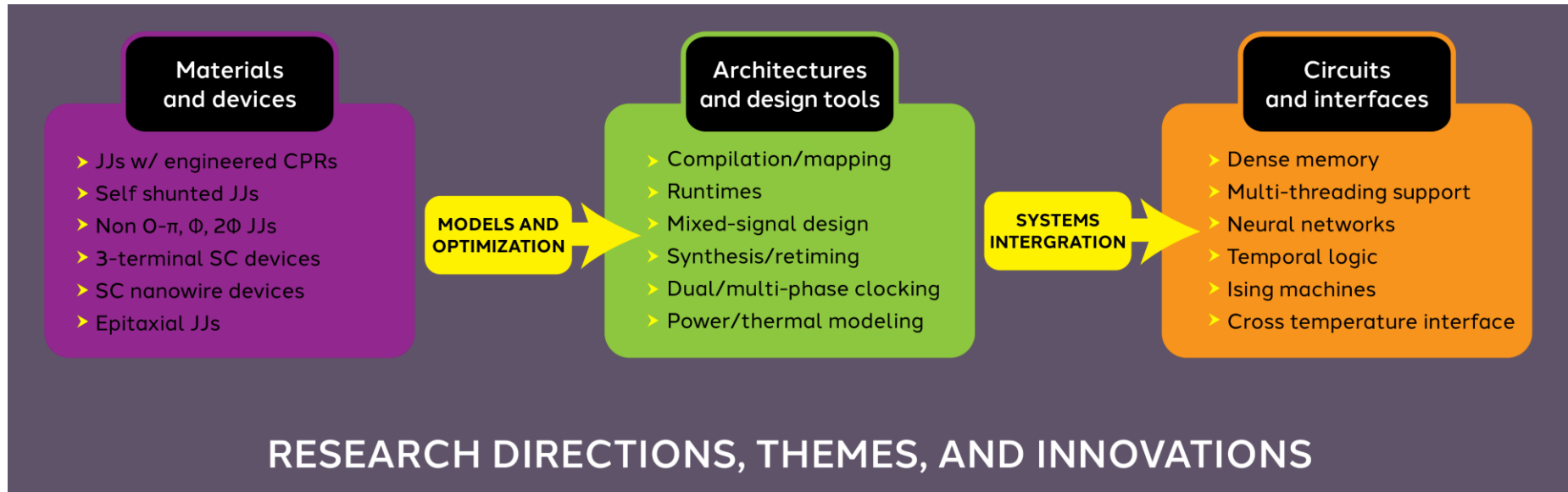
Petaflop/s-days

1e+4



NSF DISCoVER Expedition

Design & Integration of Superconductive Computation For Ventures beyond Exascale Realization



OUTLINE

Background

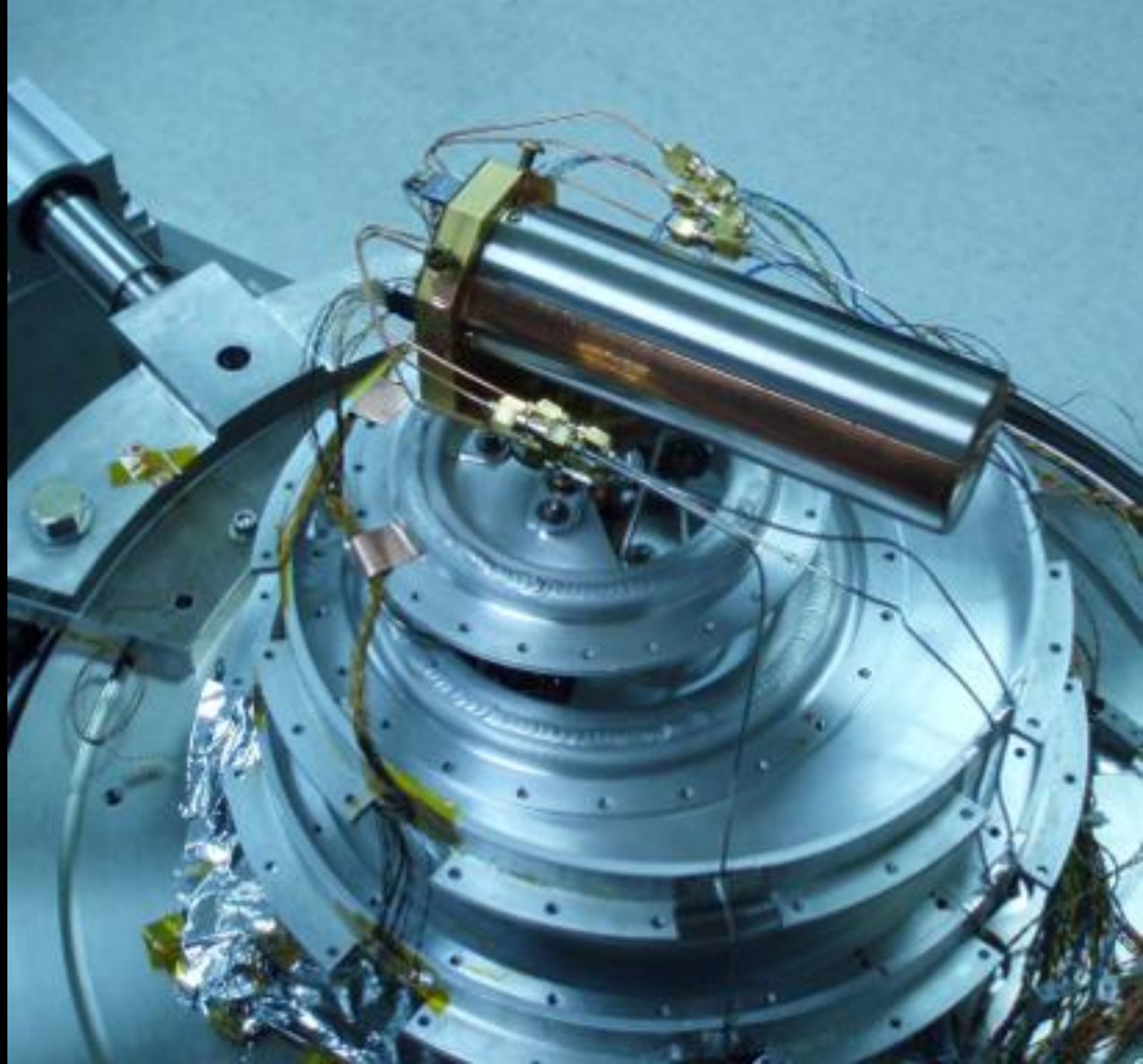
SFQ technology
SFQ Logic and Circuits



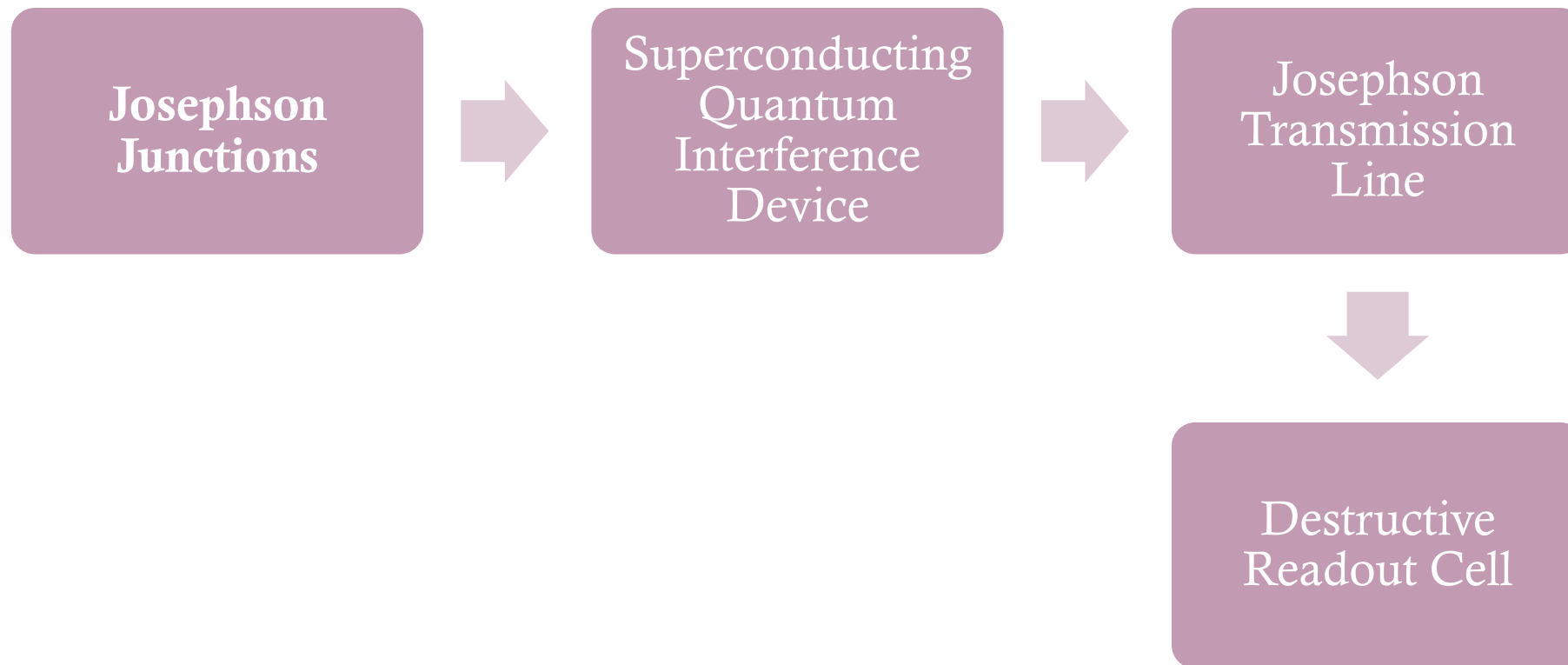
DISCOVER
CPU
Architecture

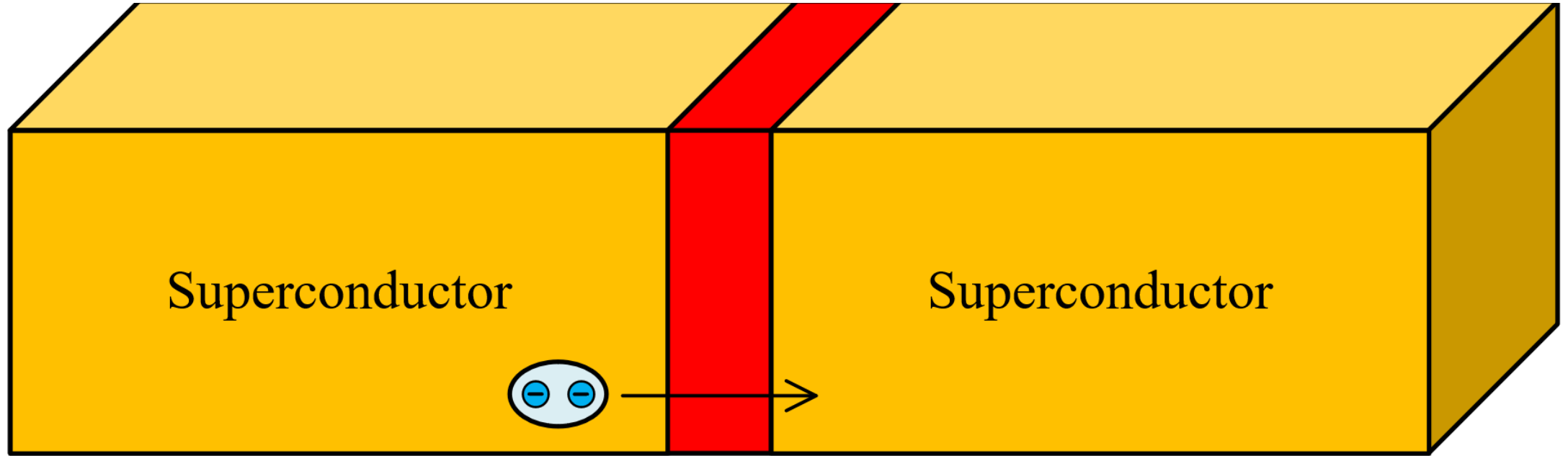
Multi-Fluxon Storage Design
Multi-Fluxon Register File
And more..

SFQ TECHNOLOGY



SFQ TECHNOLOGY



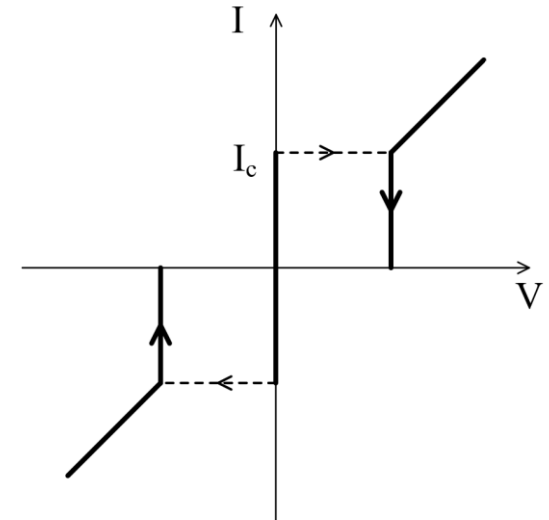
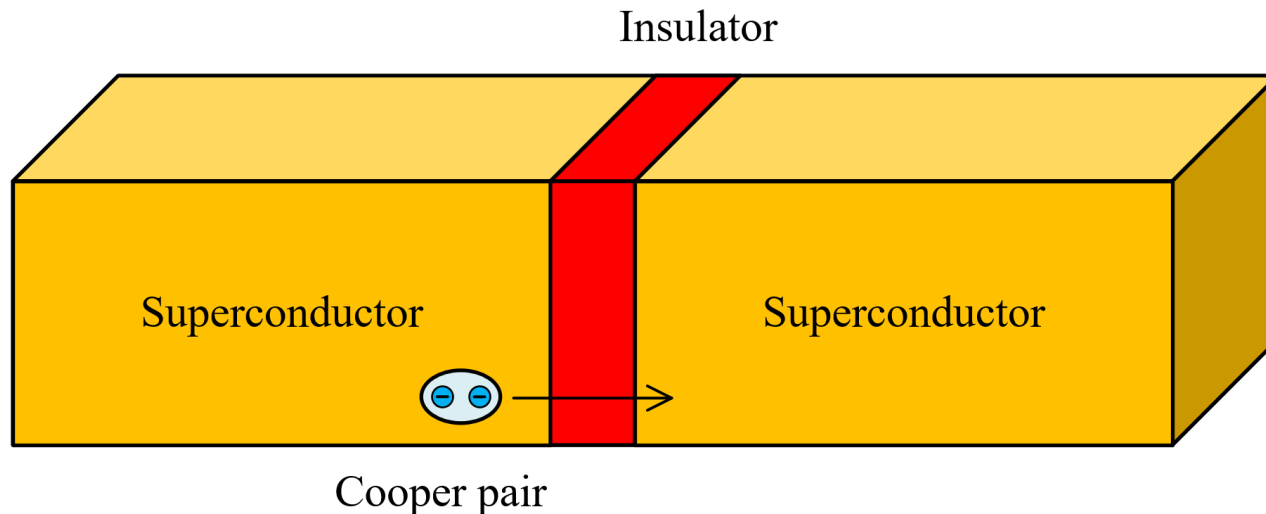


JOSEPHSON JUNCTIONS

- Single Flux Quantum(SFQ) technology uses superconducting devices, namely Josephson Junctions(JJ), to build traditional digital circuits
- JJs work at a low temperature with a short switching time ($\sim 1\text{ps}$) and little switching energy dissipation ($\sim 10^{19}$ Joules)

Josephson Junctions

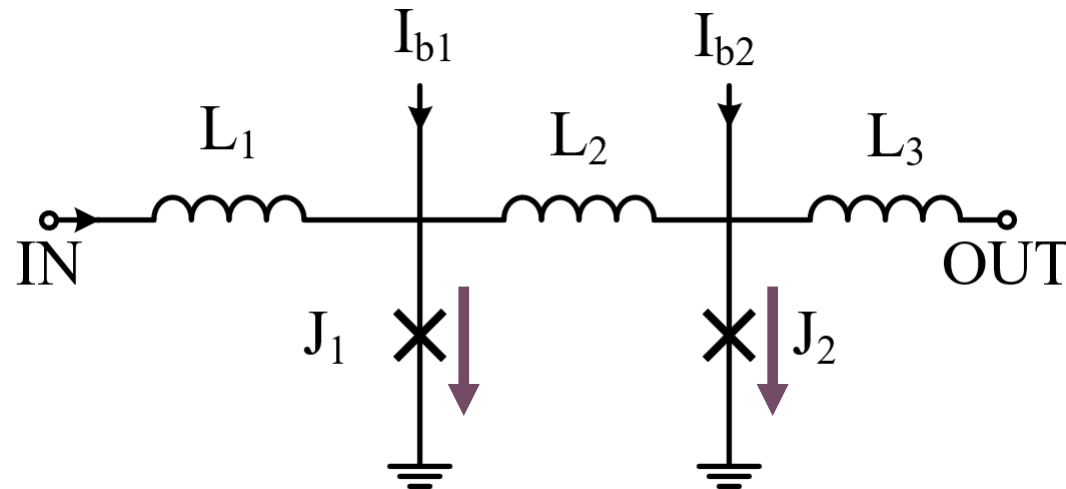
- When the cooper pair current is small ($I < I_c$), JJ behaves like an ordinary superconductor
- When the current is large ($I > I_c$), JJ loses its superconductivity, and we call it switched



Josephson Transmission Line

Josephson Transmission Line (JTL) shows how the SFQ devices transfer the information

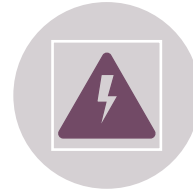
The SQUID of JTL has one stable state. In this state, the current flow through both JJs is close to their critical current



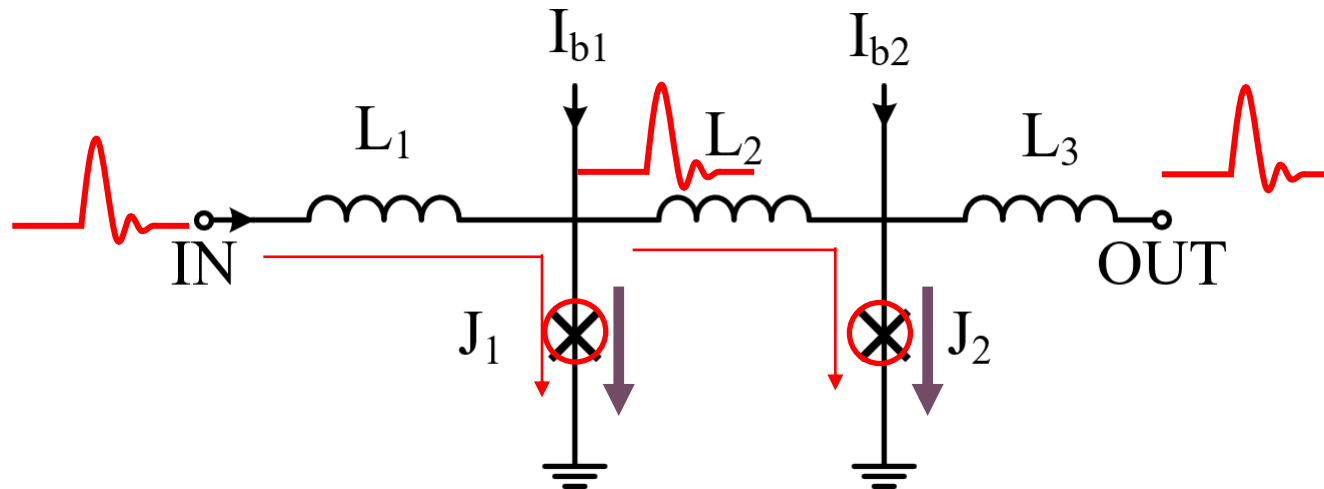
Josephson Transmission Line



An input voltage pulse causes current flow through J_1 to increase. And, eventually exceeds the critical current

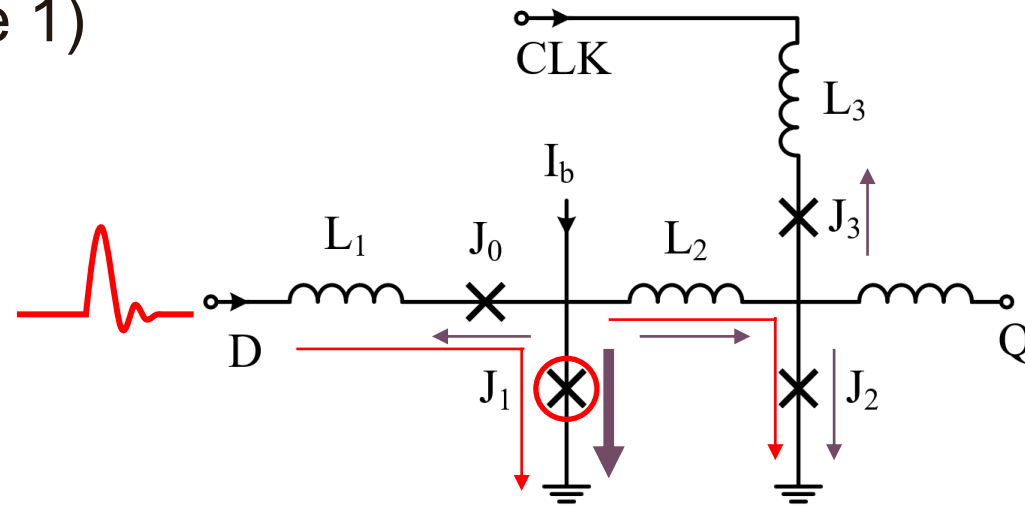


J_1 switches and creates a voltage pulse, the new pulse will increase the current flow through J_2 , then J_2 will switch and create an output voltage



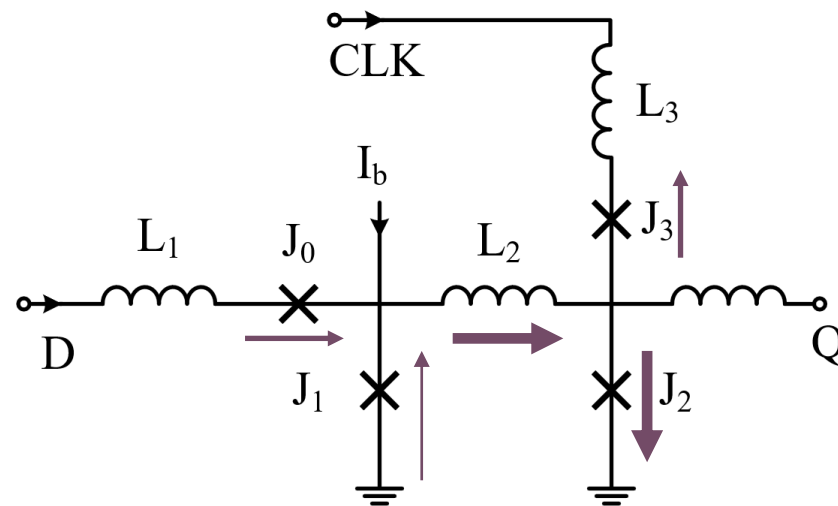
Destructive Readout Cell

- Input voltage pulse to the input \rightarrow Current through J_1 increases, and exceeds the critical current
- Then, the current flow through J_2 increases, but the current is still less than the critical current. As a result, the SQUID enters another stable state (state 1)



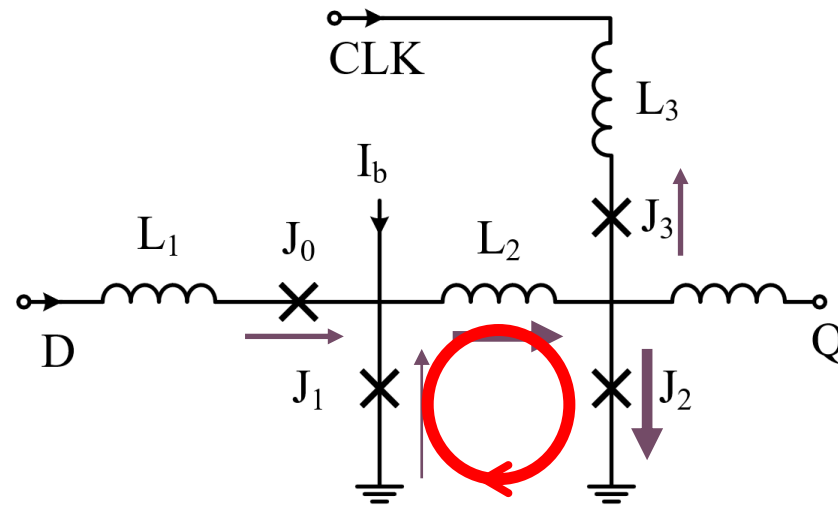
Destructive Readout Cell: Store 1

- Input voltage pulse to the input \rightarrow Current through J_1 increases, and exceeds the critical current
- Then, the current flow through J_2 increases, but the current is still less than the critical current. As a result, the SQUID enters another stable state (state 1)



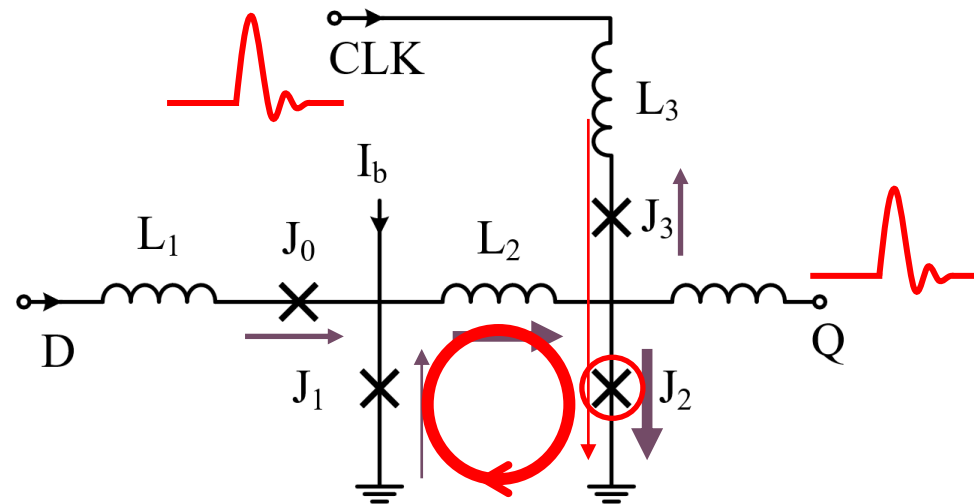
Destructive Readout Cell

- In this state, we call the SQUID trap or store the magnetic flux quanta



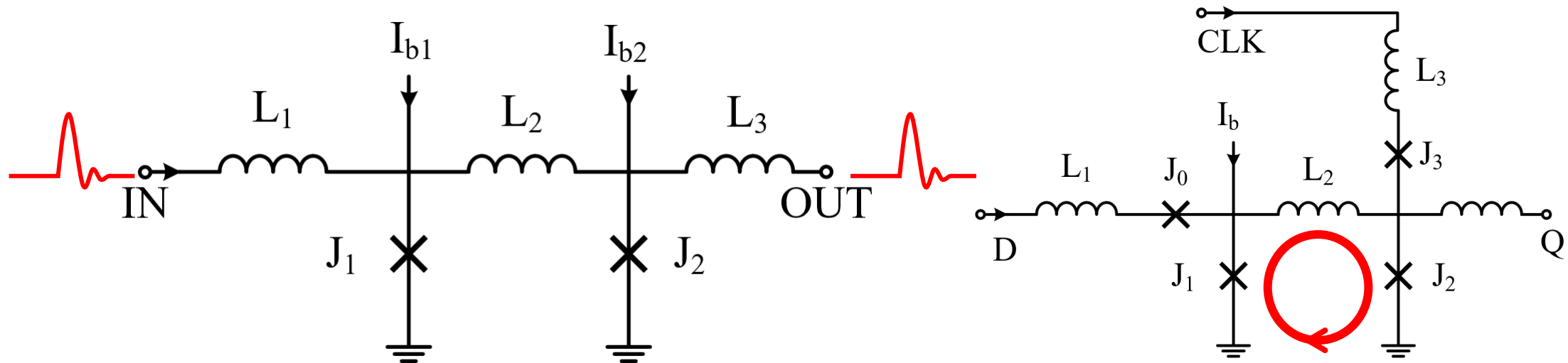
Destructive Readout Cell: Read out

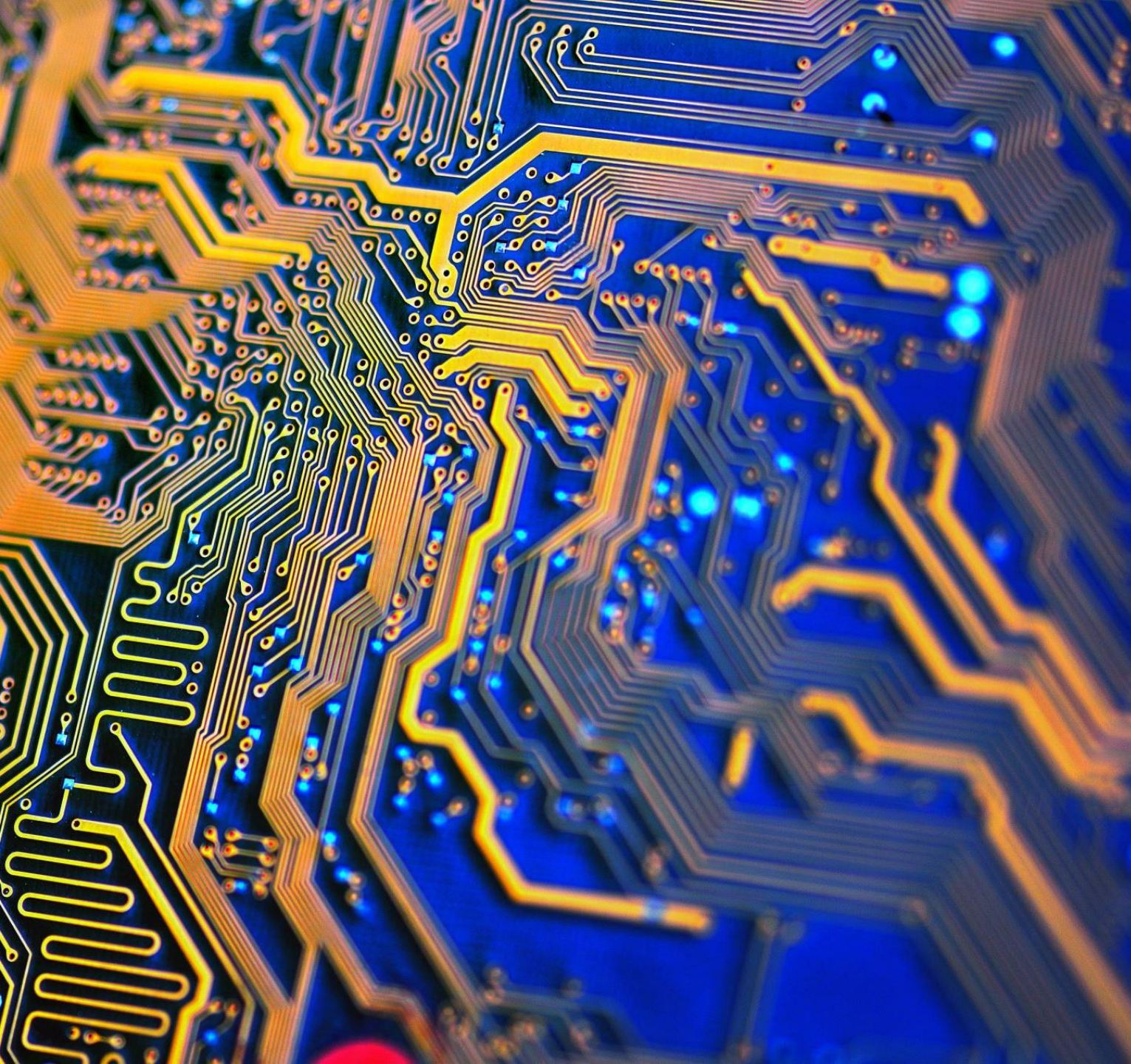
- In state 1, if we send the voltage pulse to the clock, the current flow through J_2 will exceed the critical current, and J_2 will switch. As a result, the SQUID goes back to state 0 and generates an output voltage pulse
- This is how we read the data stored in the DRO cell



SFQ Technology

- Since the information is stored as magnetic flux quanta and transferred as a Single Flux Quantum voltage pulse, the technology is called Single Flux Quantum(SFQ) technology





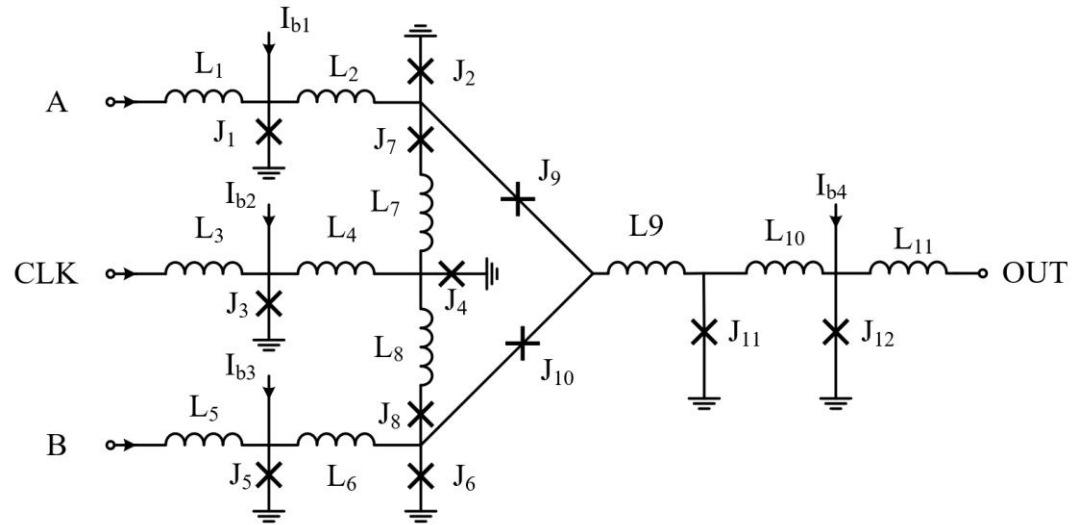
SFQ LOGIC AND CIRCUITS

SFQ LOGIC REPRESENTATION

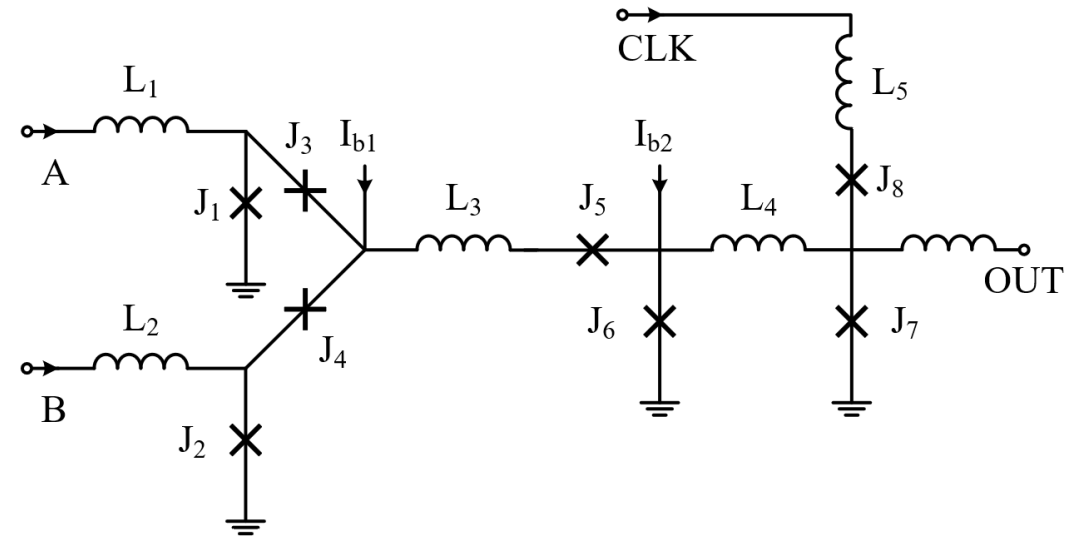
In SFQ logic, the arrival of inputs at a gate does not mean the gate will perform the function

It needs an explicit clock to trigger the pulse traversal and eventual computation

SFQ AND gate



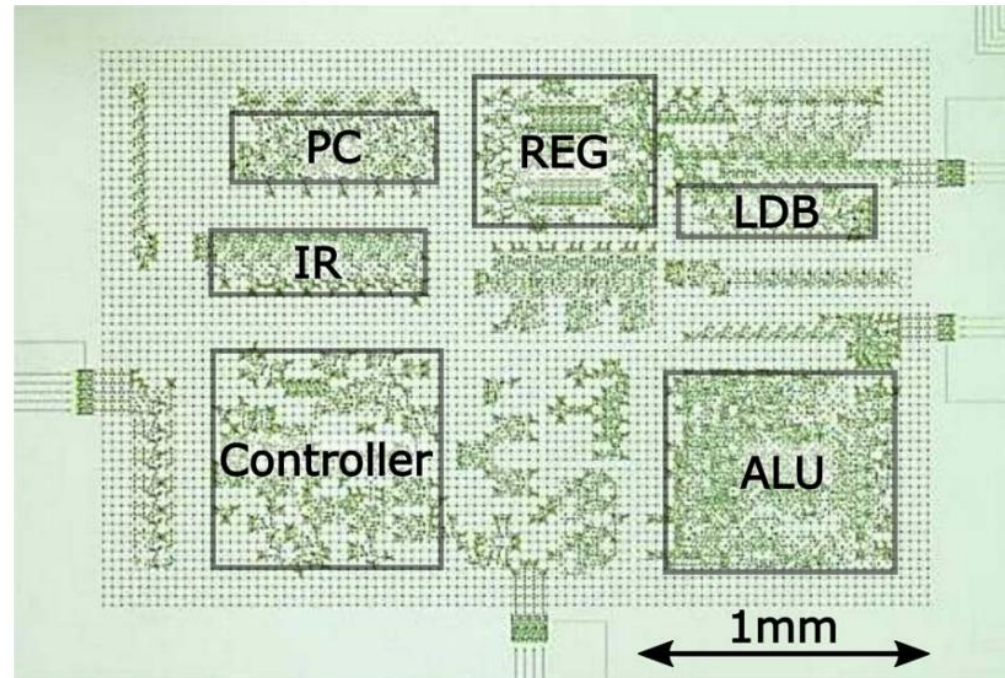
SFQ OR gate



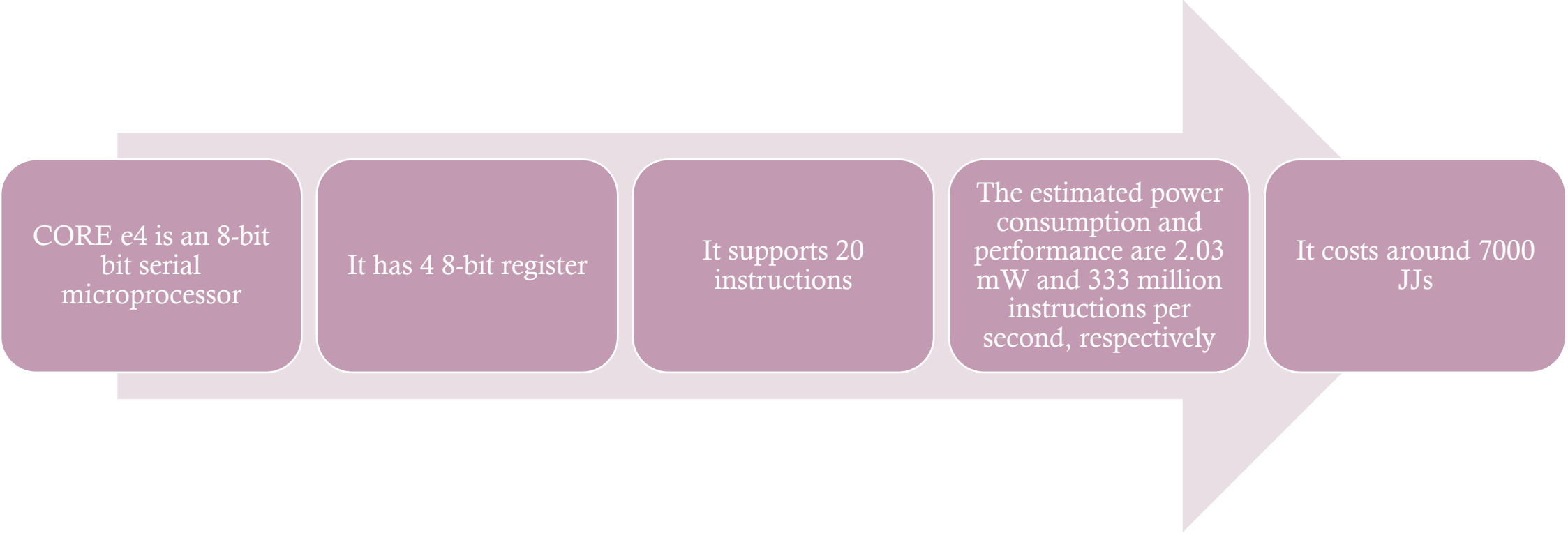
SFQ LOGIC GATES

SFQ Microprocessor

- Most of the SFQ microprocessor research focused on fabricating a tiny microprocessor or optimizing the ALU datapath



CORE E4 MICROPROCESSOR



CORE e4 is an 8-bit
bit serial
microprocessor

It has 4 8-bit register

It supports 20
instructions

The estimated power
consumption and
performance are 2.03
mW and 333 million
instructions per
second, respectively

It costs around 7000
JJs

CHALLENGES OF SFQ MICROPROCESSOR

Memory of SFQ microprocessor is expensive. The basic DRO cells can only be read once. NDRO (non-destructive) cells cost much more JJs

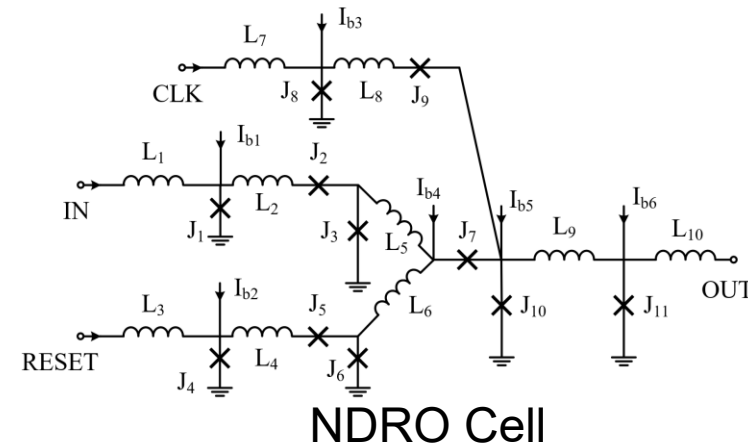
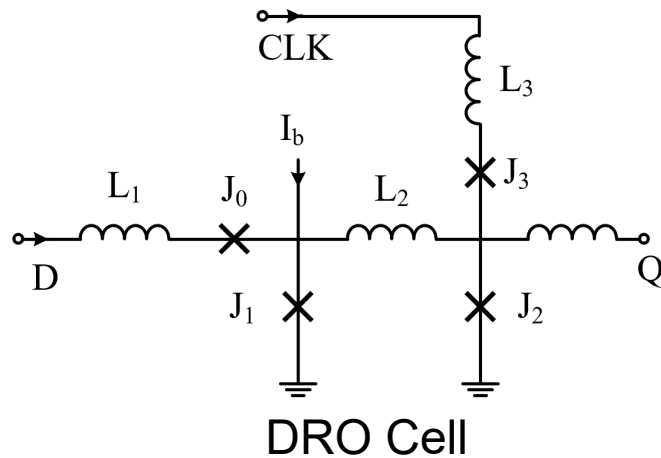
Deep gate-level pipeline makes the design of complex architectural blocks more difficult

Multi-Fluxon Storage

Katam, N. K., Zha, H., Pedram, M., & Annavaram, M. (2020, June). Multi Fluxon Storage and its Implications for Microprocessor Design. In Journal of Physics: Conference Series (Vol. 1559, No. 1, p. 012004). IOP Publishing.

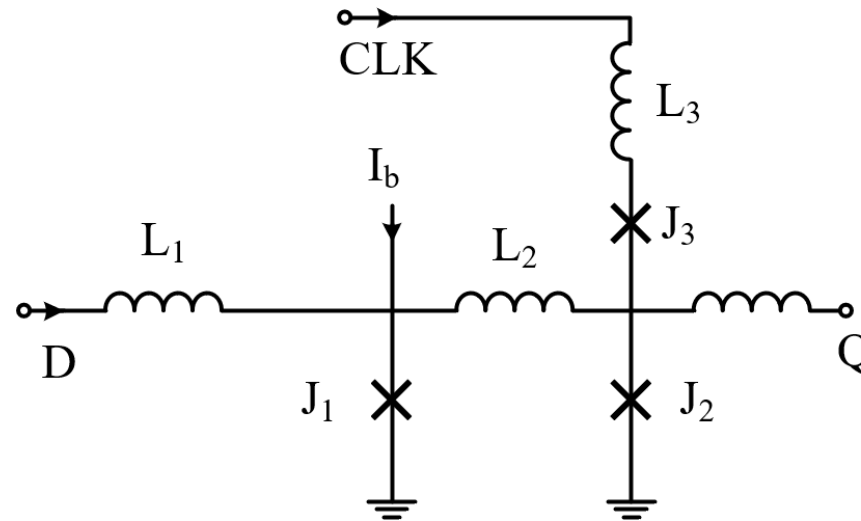
SFQ Memory Cells

- SFQ has two types of memory cells, destructive readout(DRO) and non-destructive readout(NDRO)
- By default, the pulse stored in the memory cell will be released after readout, so extra hardware is needed to retain the pulse. As such, NDRO cells are usually 3X larger than DRO cells



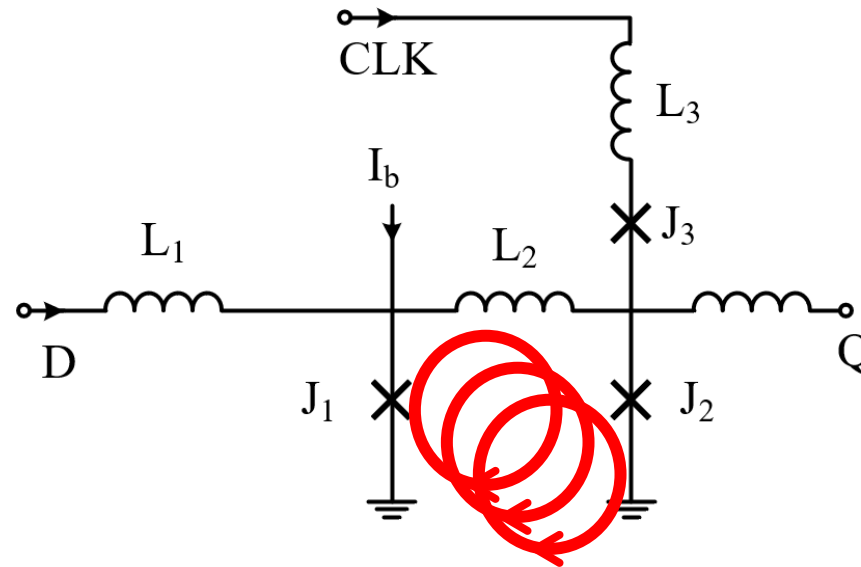
Our High-Capacity DRO(HC-DRO) Design

- J_0 prevents the accumulation of pulses
- Enable SQUID to accept multiple inputs \rightarrow we need to remove J_0
- To make sure the J_1 - L_2 - J_2 SQUID can have more stable states, we need to change their sizes



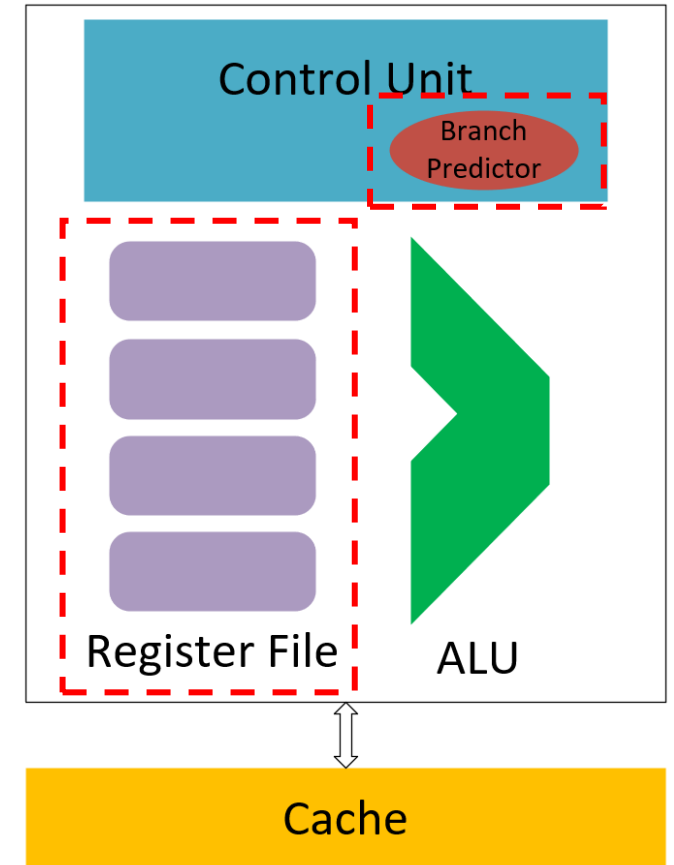
New High-Capacity DRO(HC-DRO) cell

- By having different sizes, J_1 - L_2 - J_2 SQUID can have 3 or 4 stable states, or we can say they can store at most 2 or 3 fluxons
- In the rest of the discussion, we only focus on the one can hold at most 3 fluxons



HC-DRO in Complex SFQ CPU

- The main units in the CPU that require a large amount of memory are register file, branch predictor, and cache
- Example design: register file



HiPerRF: HC-DRO Register File

H. Zha, N. K. Katam, M. Pedram, and M. Annavaram, “HiPerRF: A dual-bit dense storage sfq register file,” in 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2022.11

REGISTER FILE WITH (HC)DRO

HC-DRO has 2X capacity

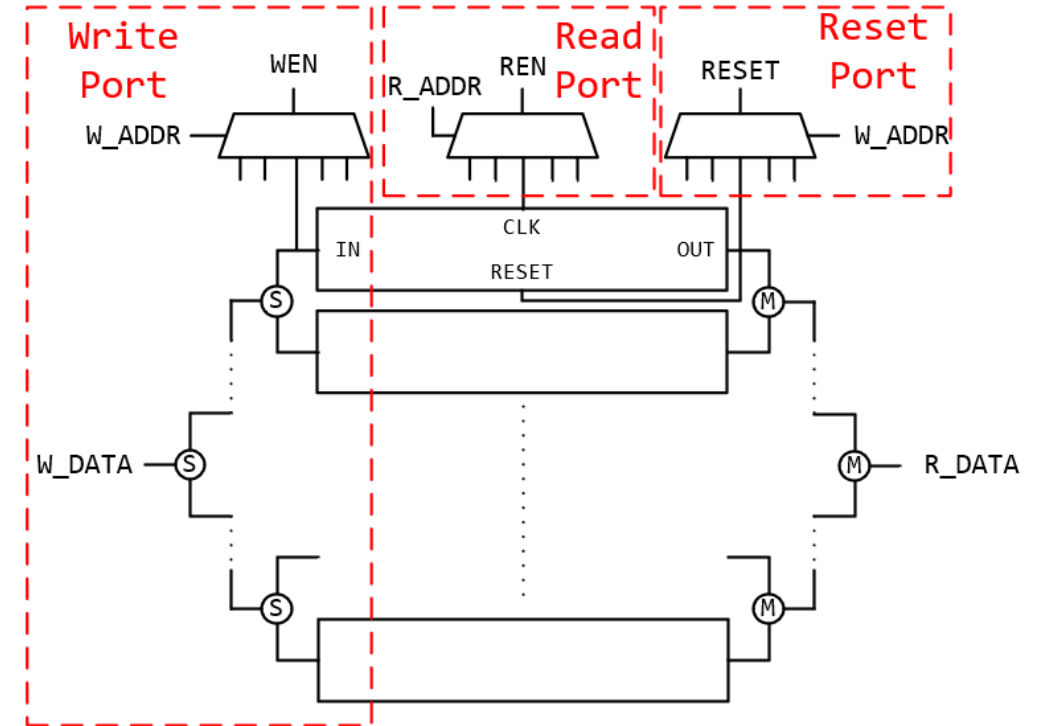
But reads are destructive

How to support multiple read outs?

- Look at victim cache for inspiration.
-

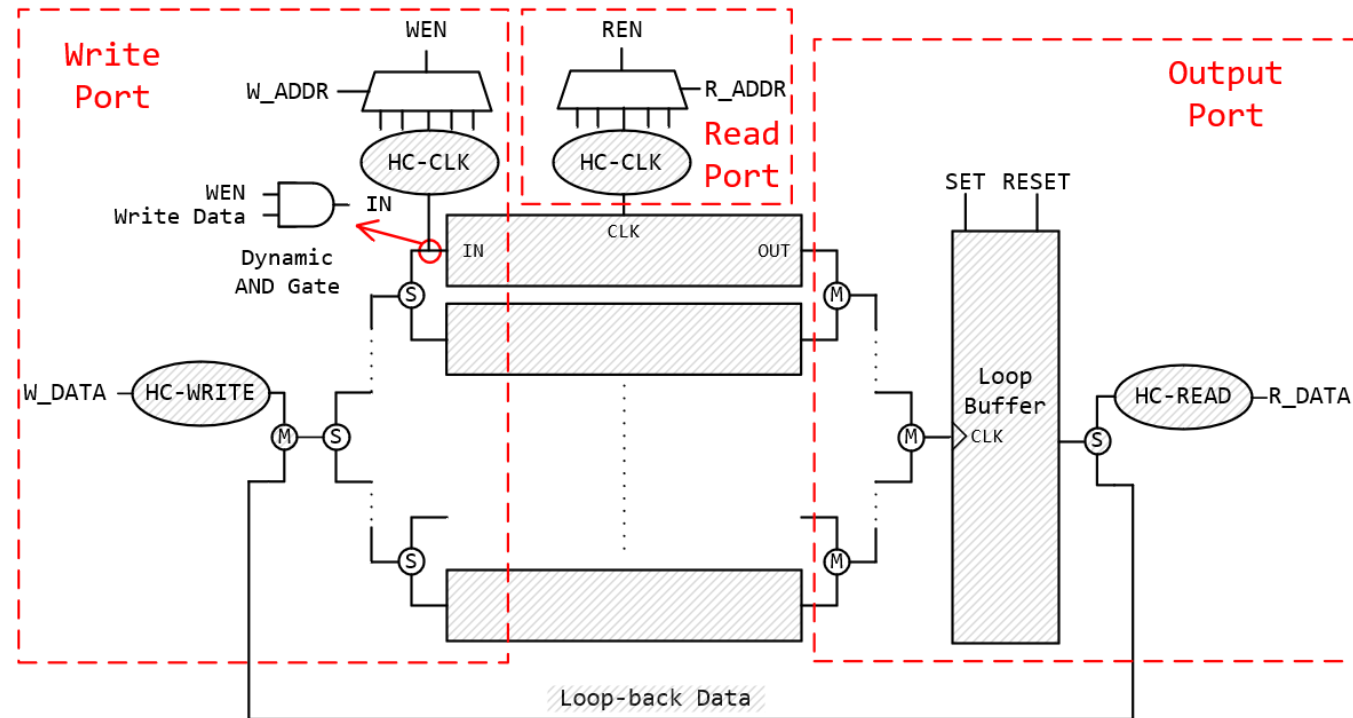
NDRO Register File Baseline

- **Two differences**
 - Read port before registers
 - CMOS DFFs always provide output passively
 - NDRO cells need to be readout actively
 - Existence of a reset port
 - A CMOS DFF storing a “1” can be overwritten with a “0”
 - A NDRO cell storing a “1” can only be reset rather than be overwritten



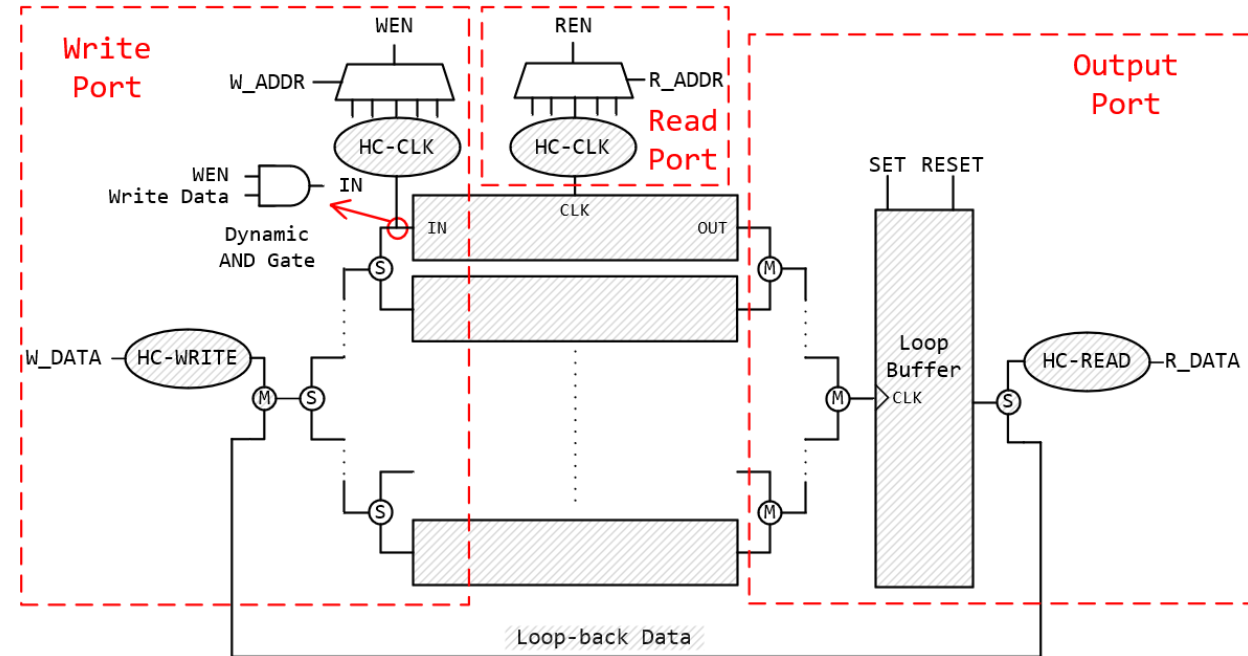
HC-DRO Register File(HiPerRF) Design

- HiPerRF uses HC-DRO cells to replace the NDRO cells to improve register file density

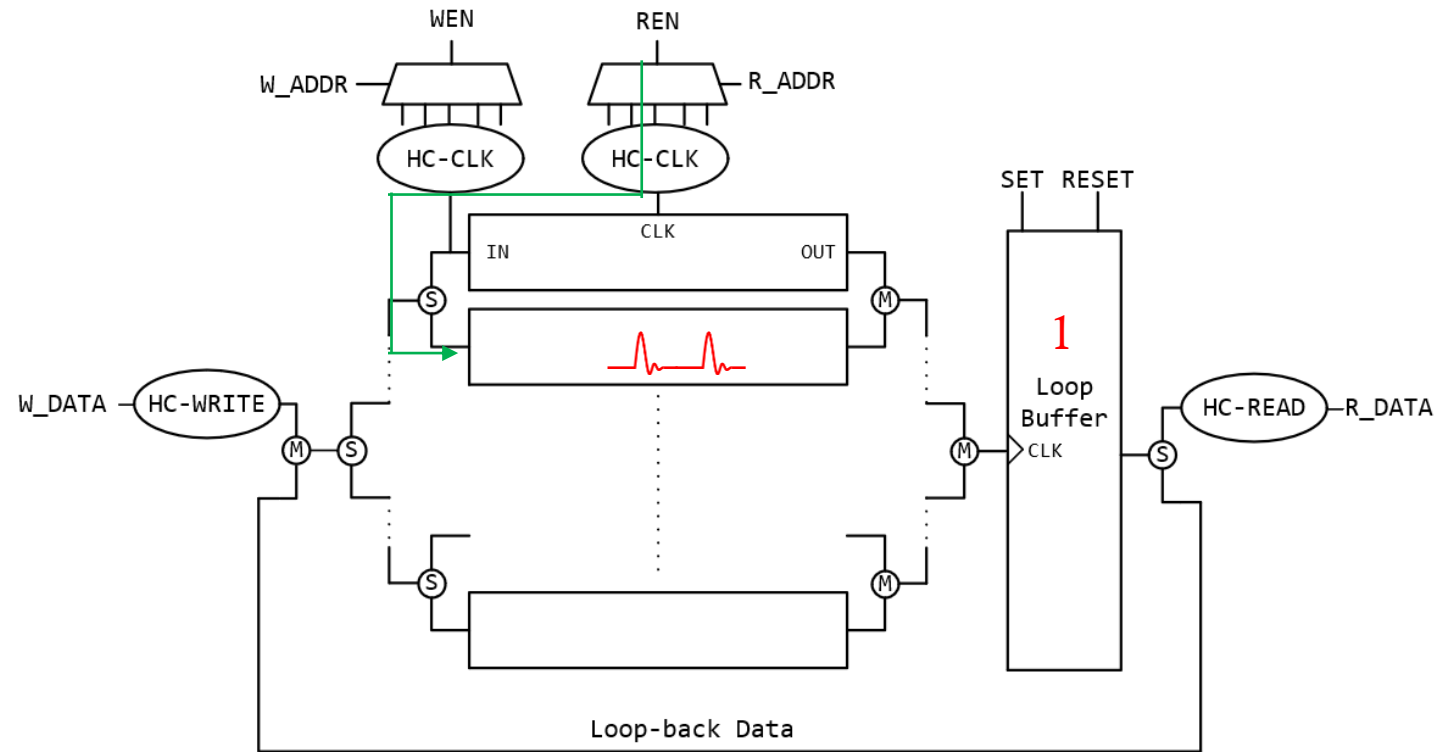


What is HiPerRF?

- Replacement of NDRO cells with HC-DRO cells
- Adding the HC-WRITE, HC-READ, and HC-CLK circuits
- The absence of a reset port
- Loop Buffer design

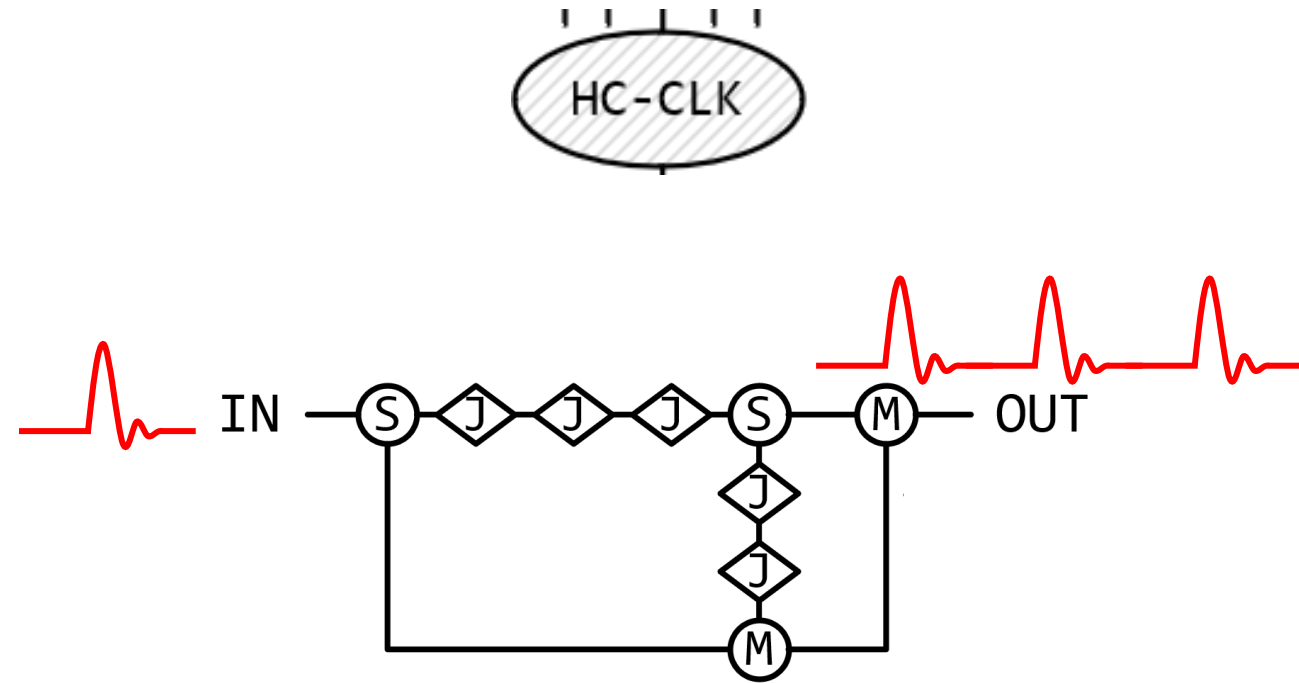


HiPerRF Read operation

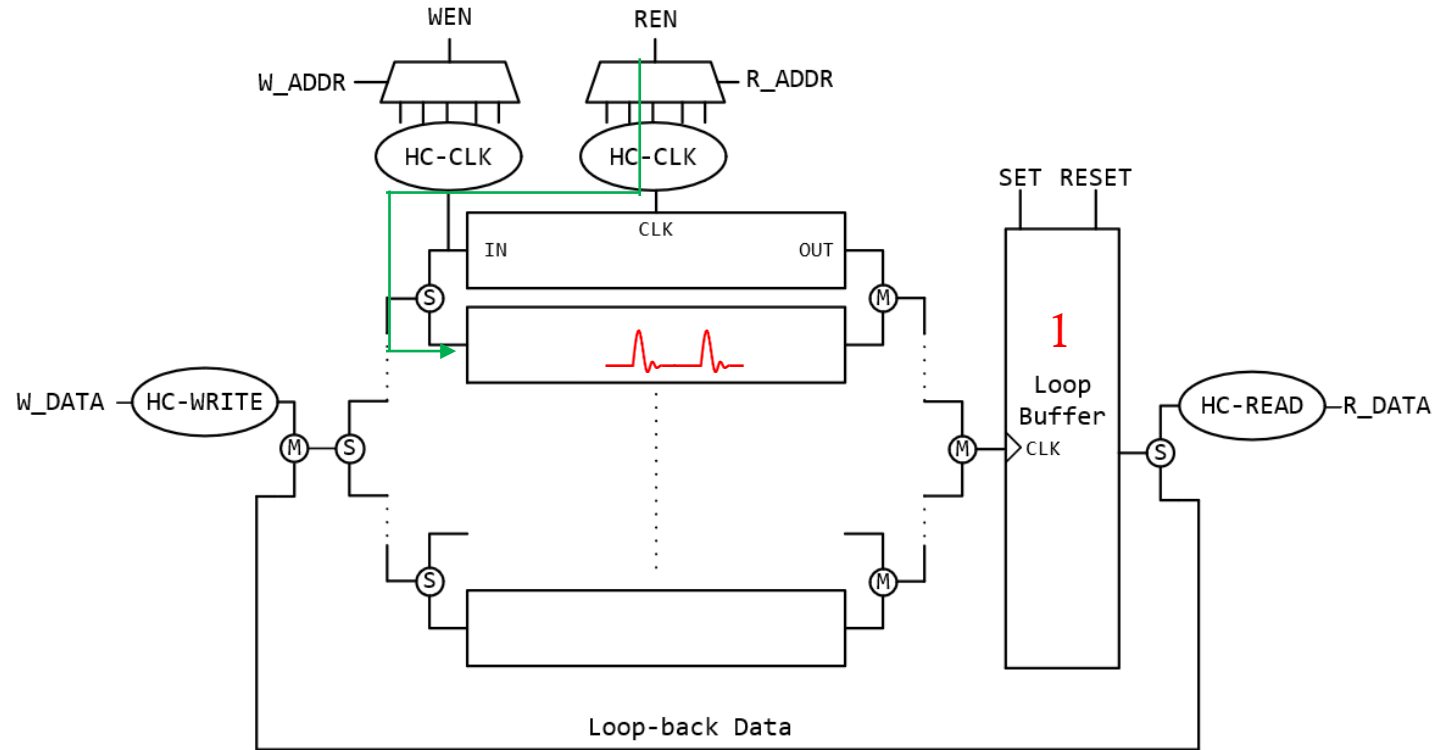


HC-CLK circuits

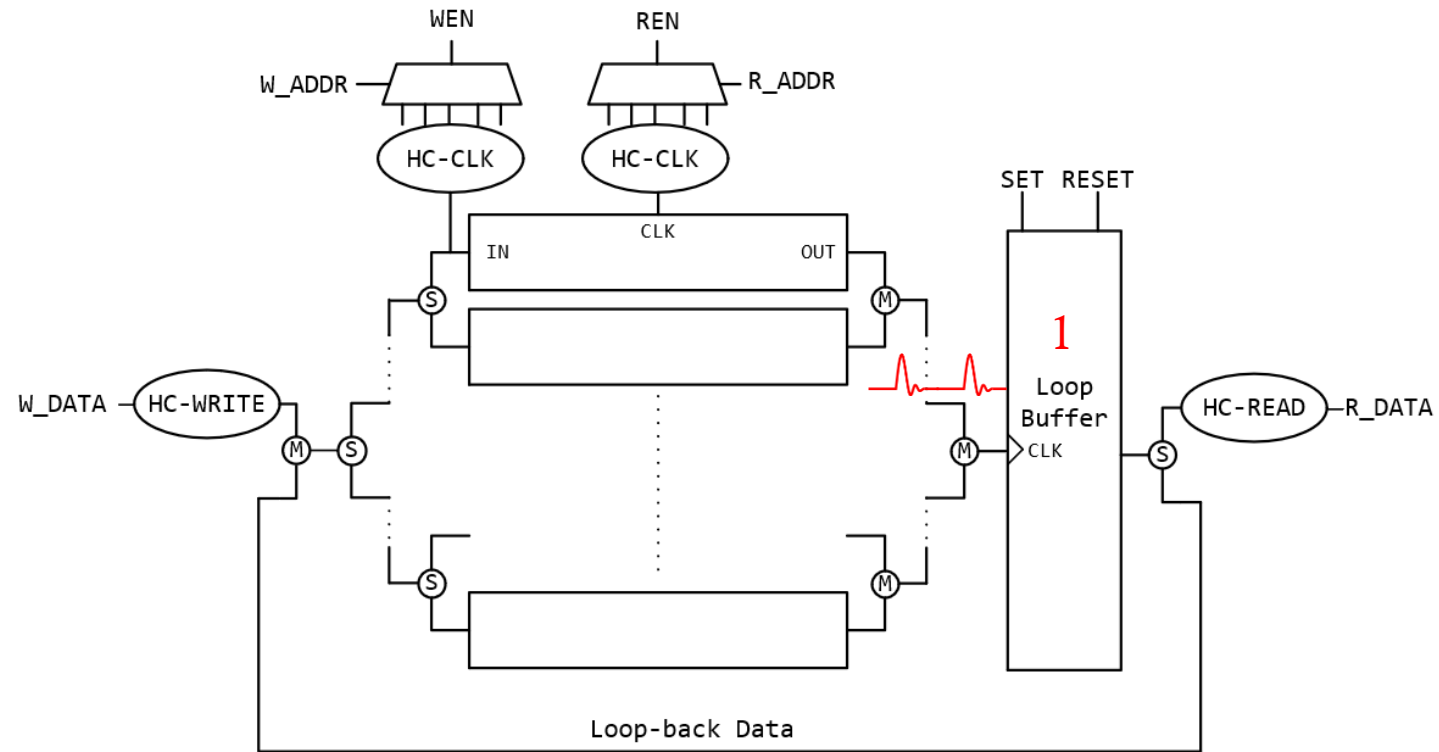
- To read HC-DRO cells, we need to send three consecutive pulses to the input pin to read out all the fluxons stored inside



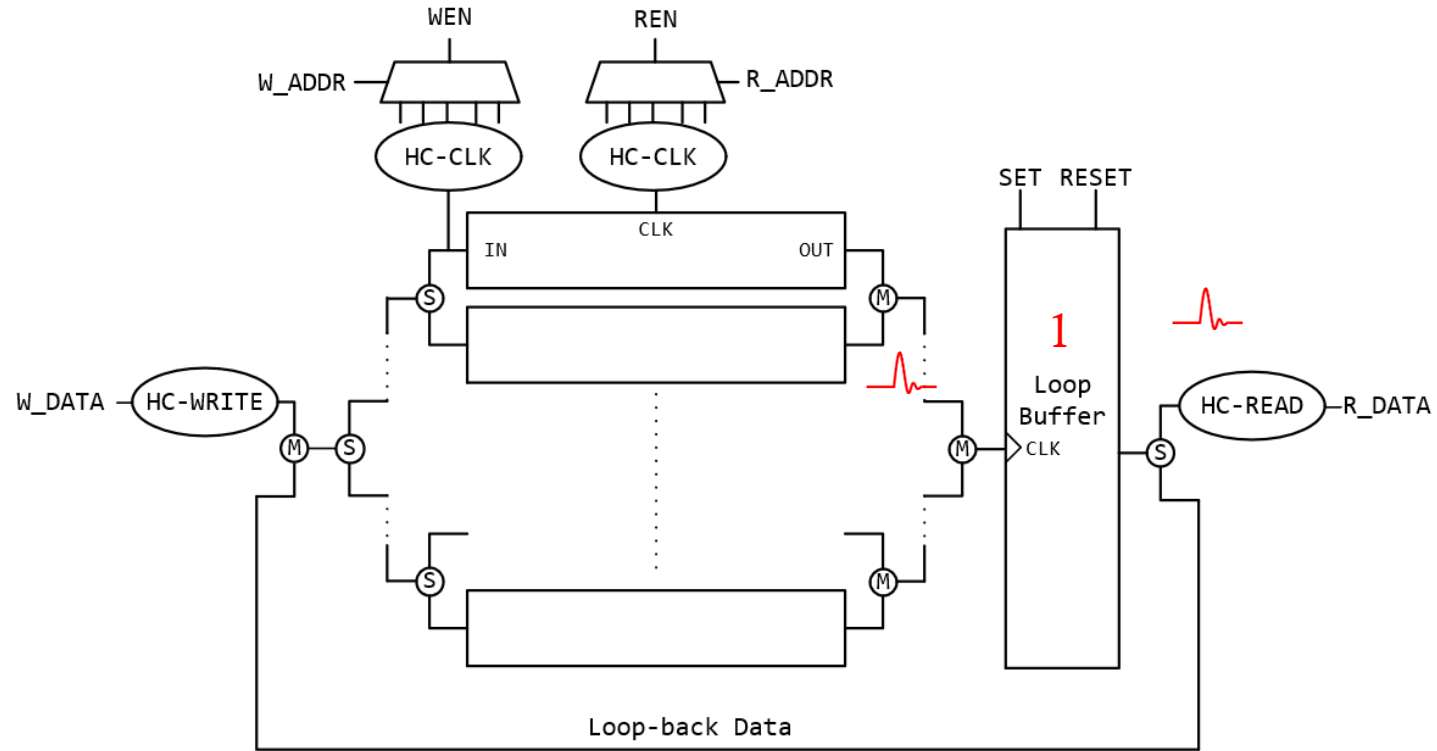
HiPerRF Read Operation



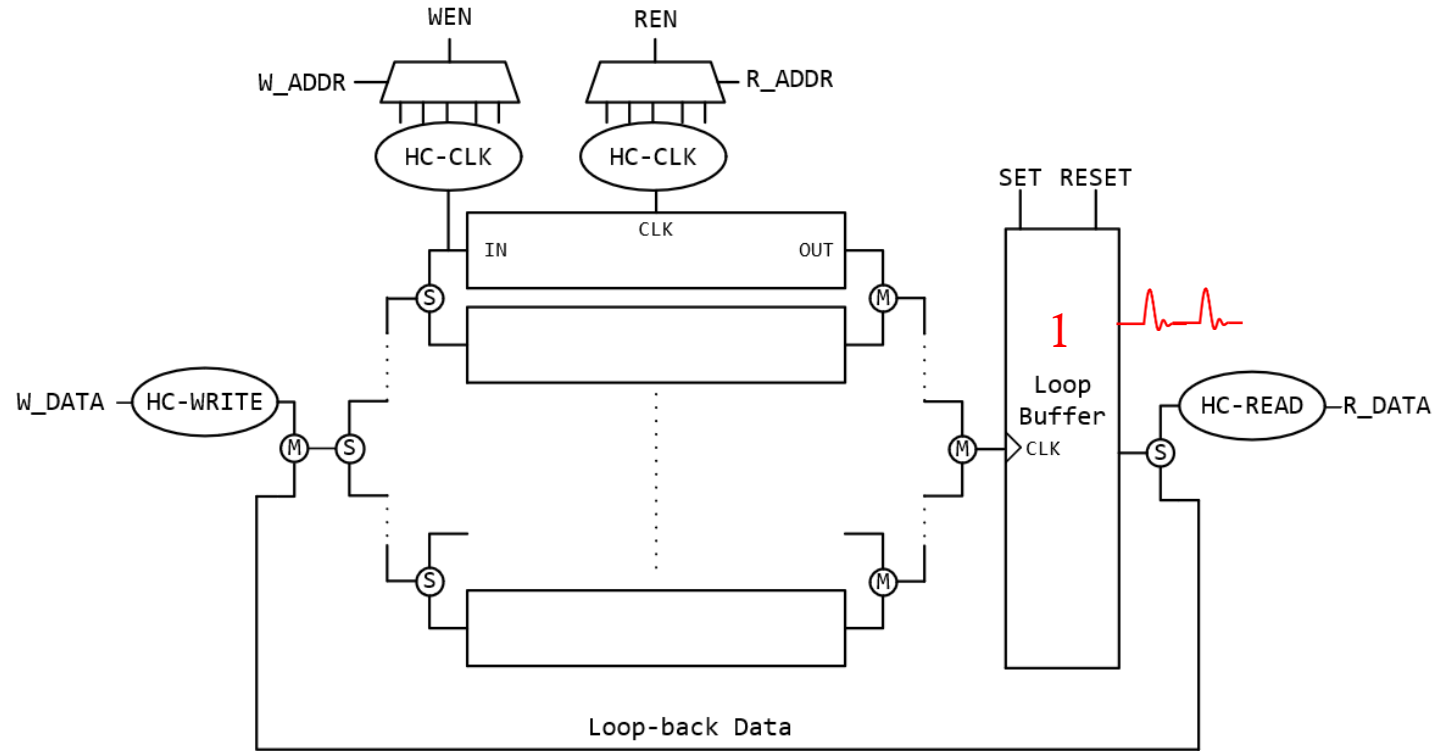
HiPerRF Read Operation



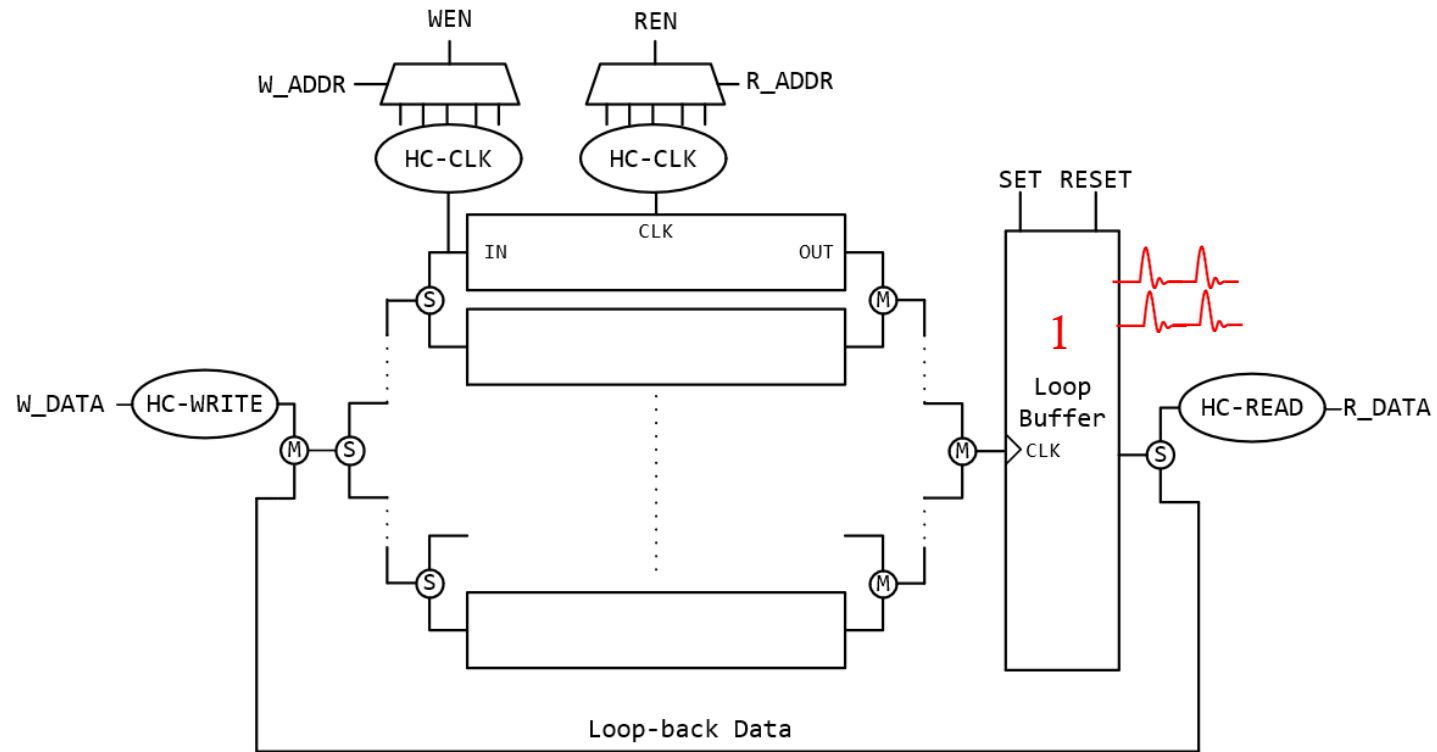
HiPerRF Read Operation



HiPerRF Read Operation



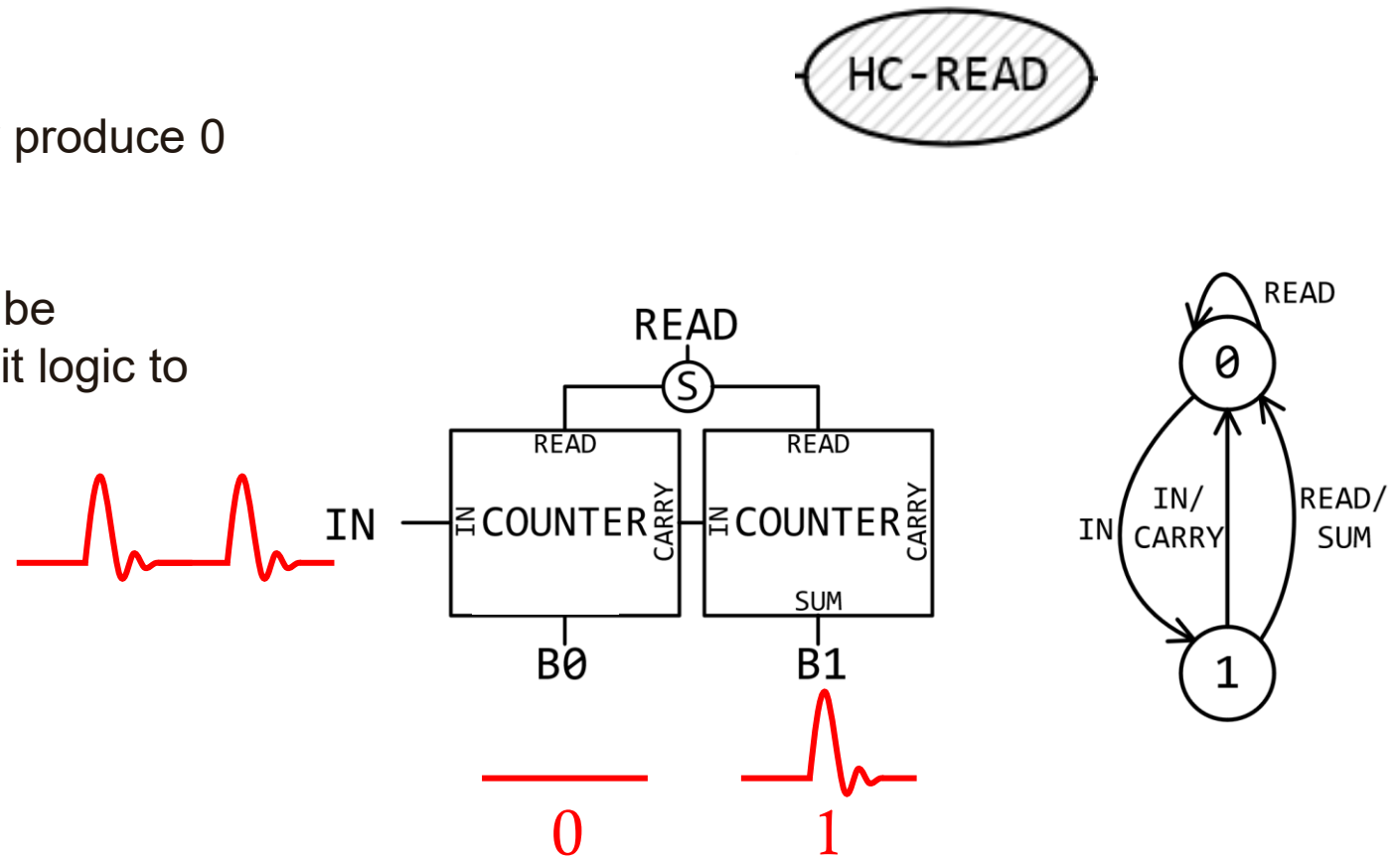
HiPerRF Read Operation



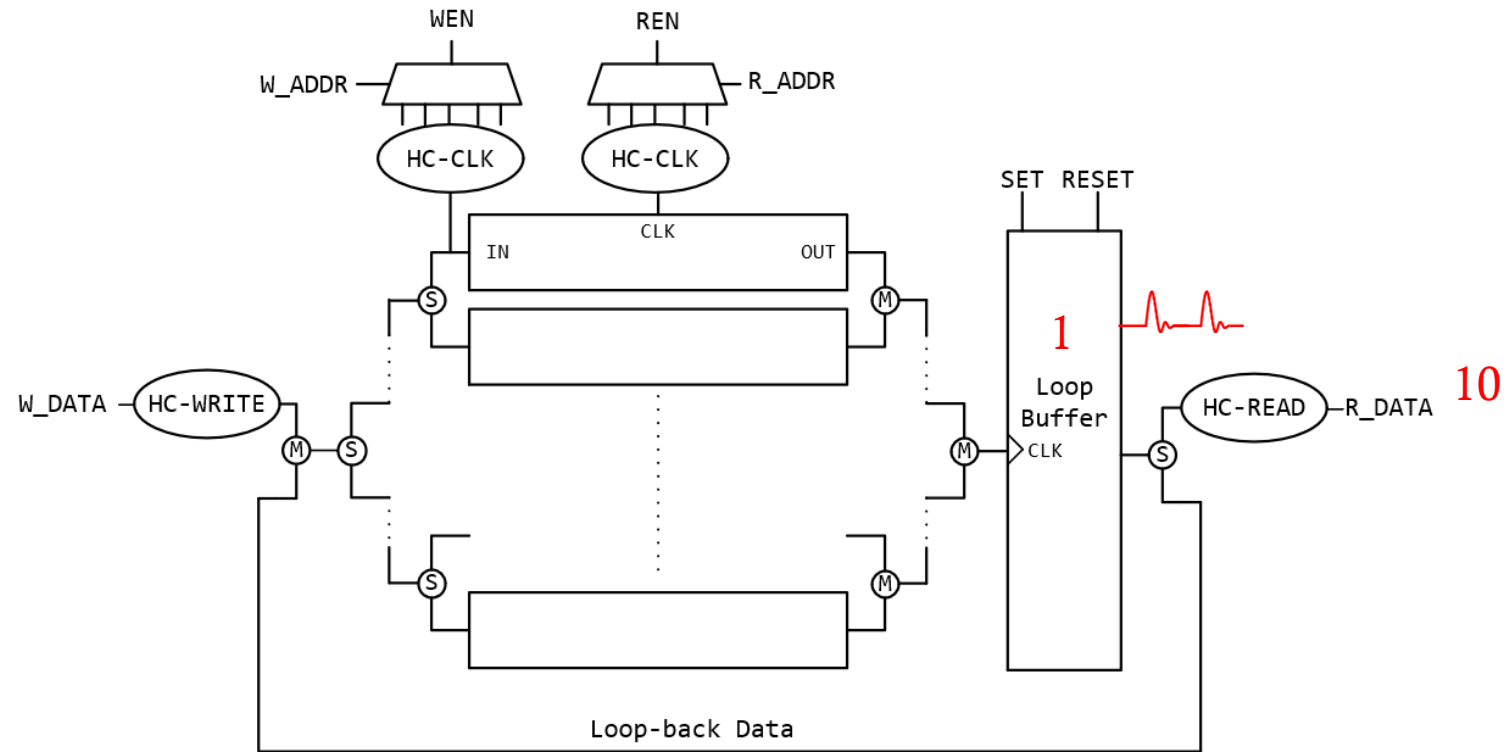
HC-READ Decoder

- HC-READ circuits

- Reading HC-DRO cells may produce 0 to 3 consecutive pulses
- These serial pulses need to be translated into normal one-bit logic to create two parallel pulses



HiPerRF Read Operation



Hardware Performance

	Total JJ Count			Percentage of Baseline		
	4 × 4	16 × 16	32 × 32	4 × 4	16 × 16	32 × 32
Register File Size (bits)						
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

Saving around 55% of JJ

TOTAL JJ COUNT AND THE PERCENTAGE OVER THE BASELINE DESIGN

Hardware Performance

Register File Size (bits)	Static Power (μ W)			Percentage of Baseline		
	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 Saving around 44% of static power

STATIC POWER AND THE PERCENTAGE OVER THE BASELINE DESIGN

Hardware Performance

Register File Size (bits)	Readout Delay (ps)			Percentage of Baseline		
	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%

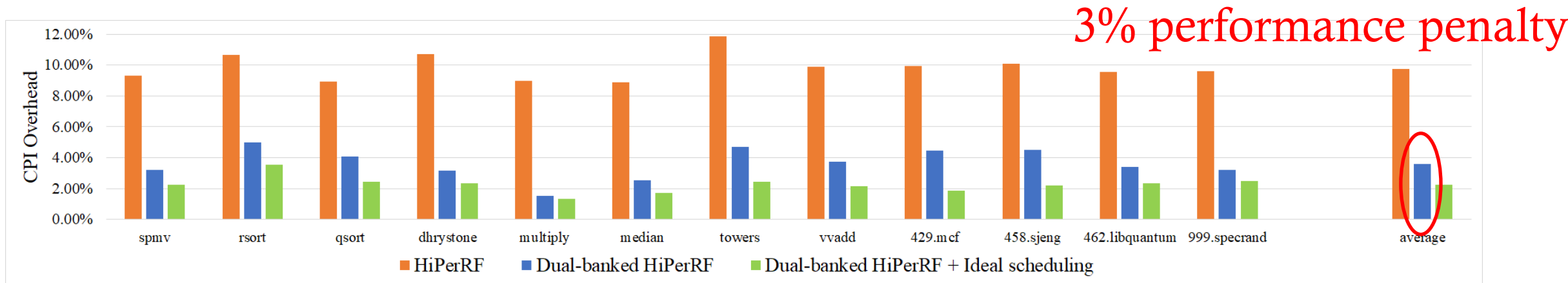
Table III

Only 8% more readout delay

READOUT DELAY AND THE PERCENTAGE OVER THE BASELINE DESIGN

Simulation Results

- We designed an SFQ based gate-level CPU simulator. The simulator is based on the RISC-V ISA Simulator Spike and written in C++.



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

SuperBP: Design Space Exploration of Perceptron-Based Branch Predictors for Superconducting CPUs

Haipeng Zha
University of Southern California
Los Angeles, USA
hzha@usc.edu

Swamit Tannu
University of Wisconsin-Madison
Madison, USA
swamit@cs.wisc.edu

Murali Annavaram
University of Southern California
Los Angeles, USA
annavara@usc.edu



<https://discoverexpedition.usc.edu/>

Why Branch Predictor?

- On-chip memory structures are JJ intensive
- Design and optimize memory-intensive microarchitecture for SFQ CPU
- Register file: HiPerRF (HPCA 2022)
- Branch predictor benefits from having more state storage

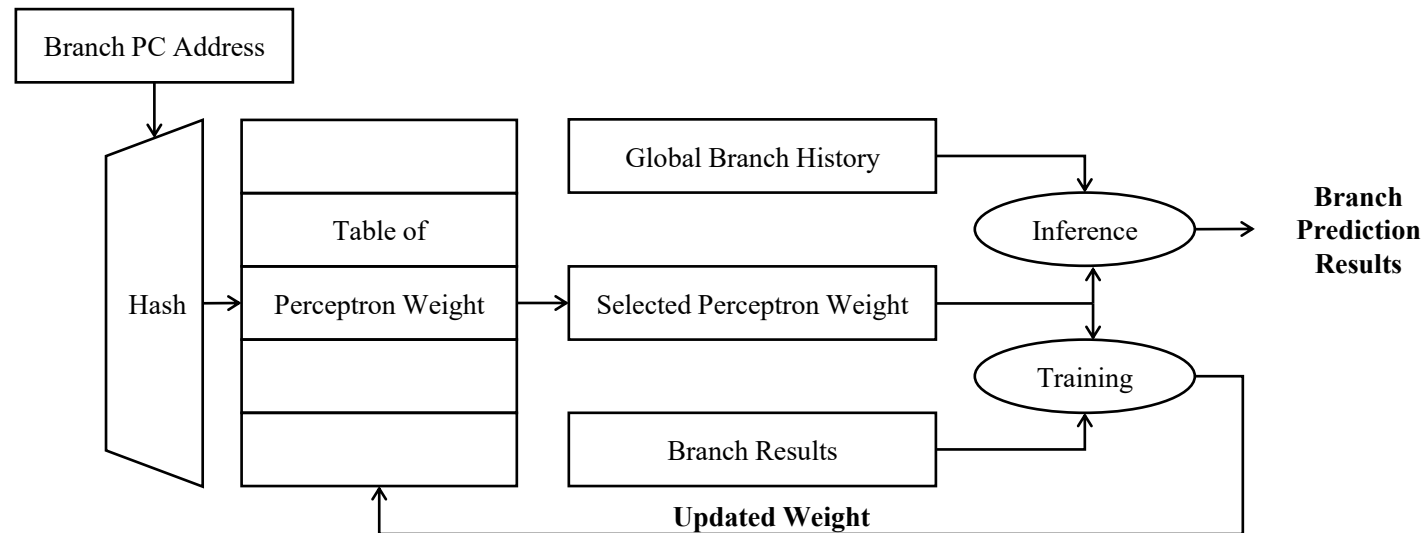
Our Goal: HC-DRO Based Branch Predictor

Outline

- **Build an SFQ perceptron branch predictor**
 - Weight Storage
 - Inference and Training Unit
- Evaluation

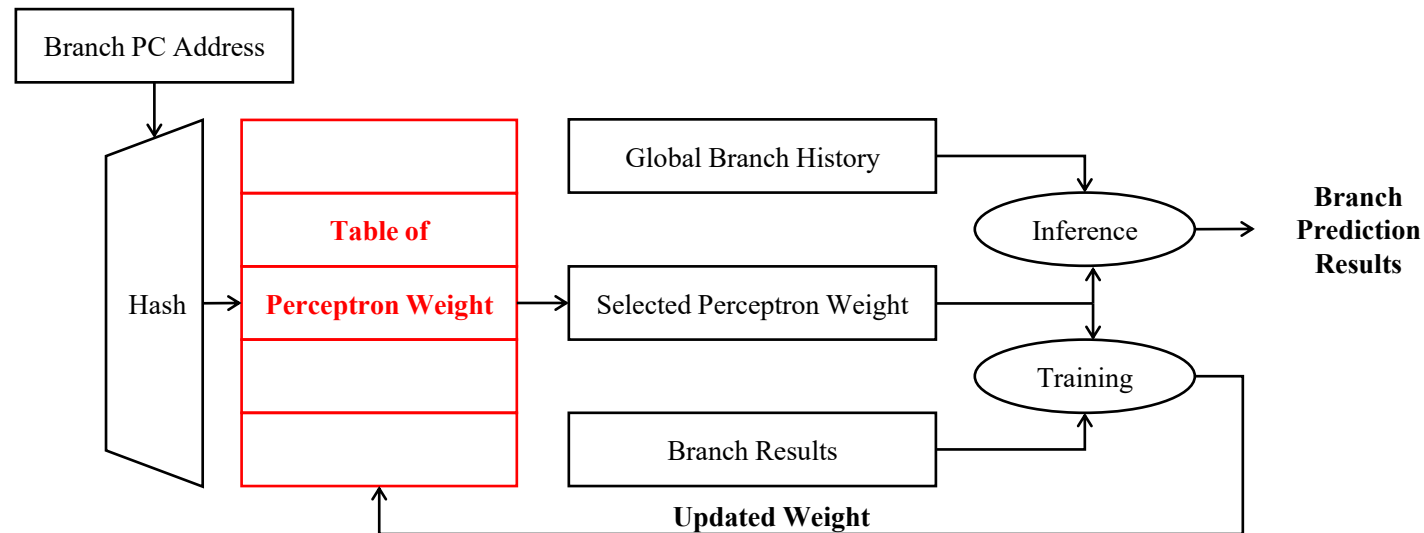
Build an SFQ Perceptron Branch Predictor

- Main units of a perceptron branch predictor



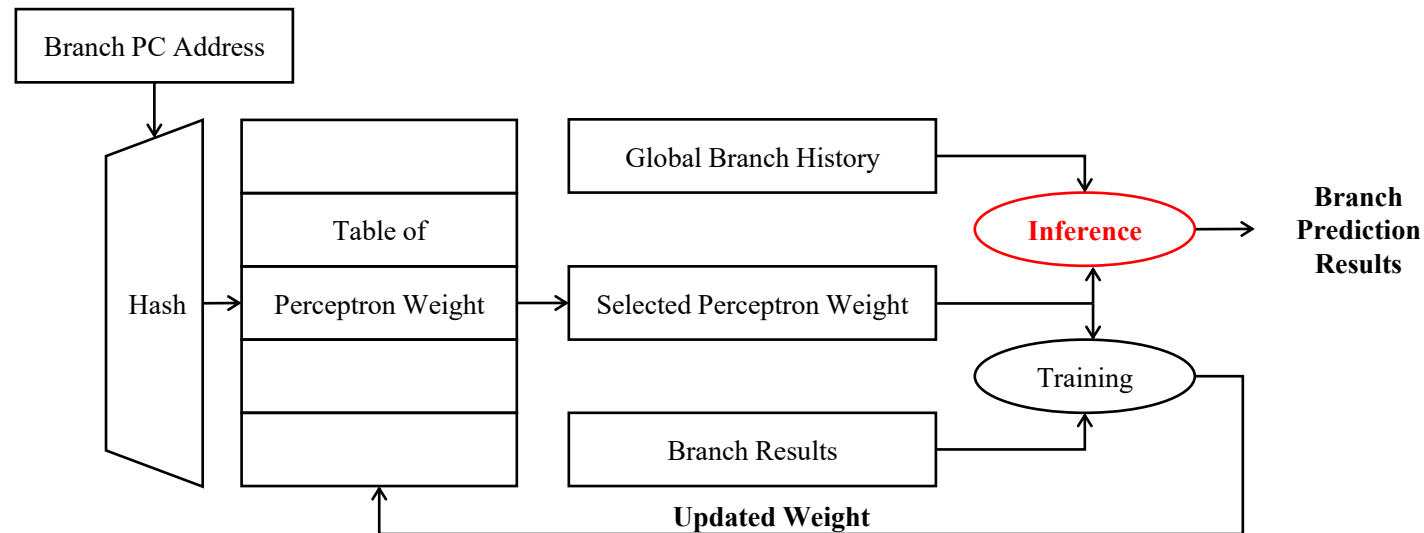
Build an SFQ Perceptron Branch Predictor

- Main units of a perceptron branch predictor
 - Perceptron weight table



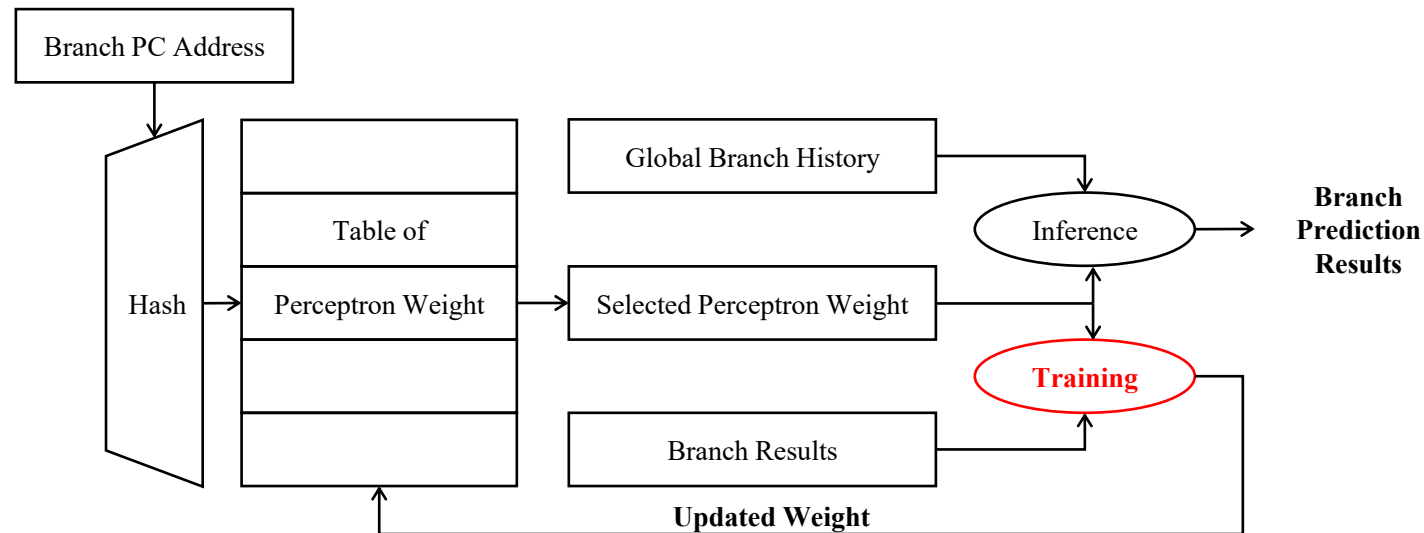
Build an SFQ Perceptron Branch Predictor

- Main units of a perceptron branch predictor
 - Perceptron weight table
 - Inference unit



Build an SFQ Perceptron Branch Predictor

- Main units of a perceptron branch predictor
 - Perceptron weight table
 - Inference unit
 - Training unit



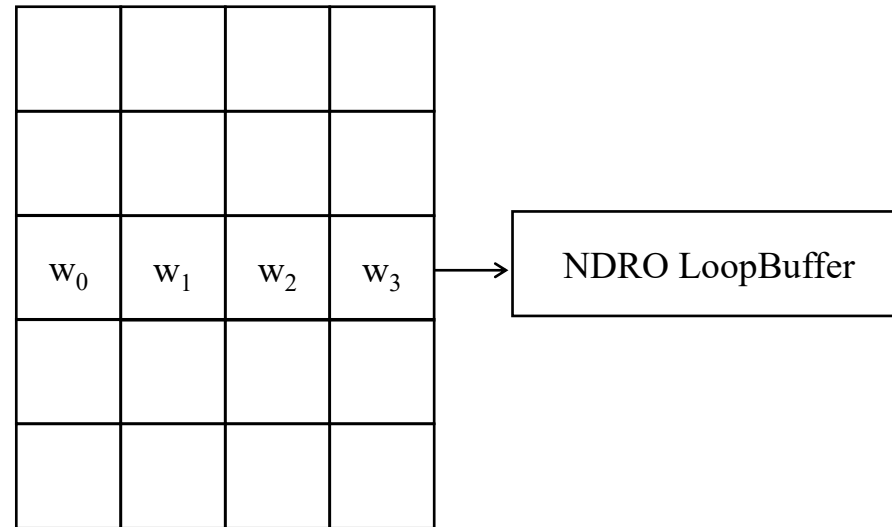
Outline

- SFQ Memory Structures
 - Build an SFQ perceptron branch predictor
 - **Weight Storage**
 - Inference and Training Unit
 - Evaluation
-

LoopBuffer for Non-destructive Readout

- We use ONE NDRO ENTRY as a loopback buffer
 - Readout data back to HC-DRO weight table

HC-DRO perceptron weight table



LoopBuffer for Non-destructive Readout

- We use ONE NDRO ENTRY as a loopback buffer
 - Readout data back to HC-DRO weight table
 - We solved the destructive readout issue

HC-DRO perceptron weight table

w_0	w_1	w_2	w_3

w_2 

w_2

NDRO LoopBuffer

Inference Unit

Outline

- SFQ Memory Structures
 - Build an SFQ perceptron branch predictor
 - Weight Storage
 - **Inference and Training Unit**
 - Evaluation
-

Inference and Training Unit Design

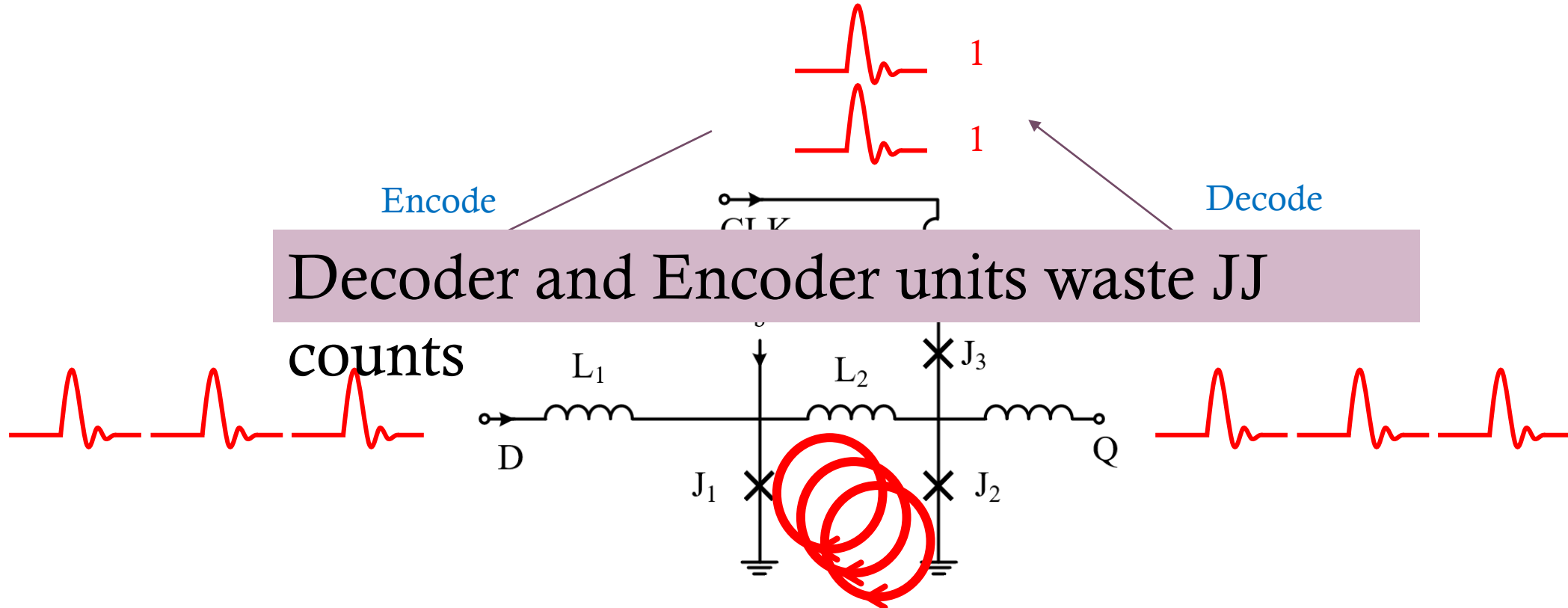
- But the design is non-trivial
- Why?

Inference and Training Unit Design

- But the design is non-trivial
- Why?
 - Need decoding and encoding

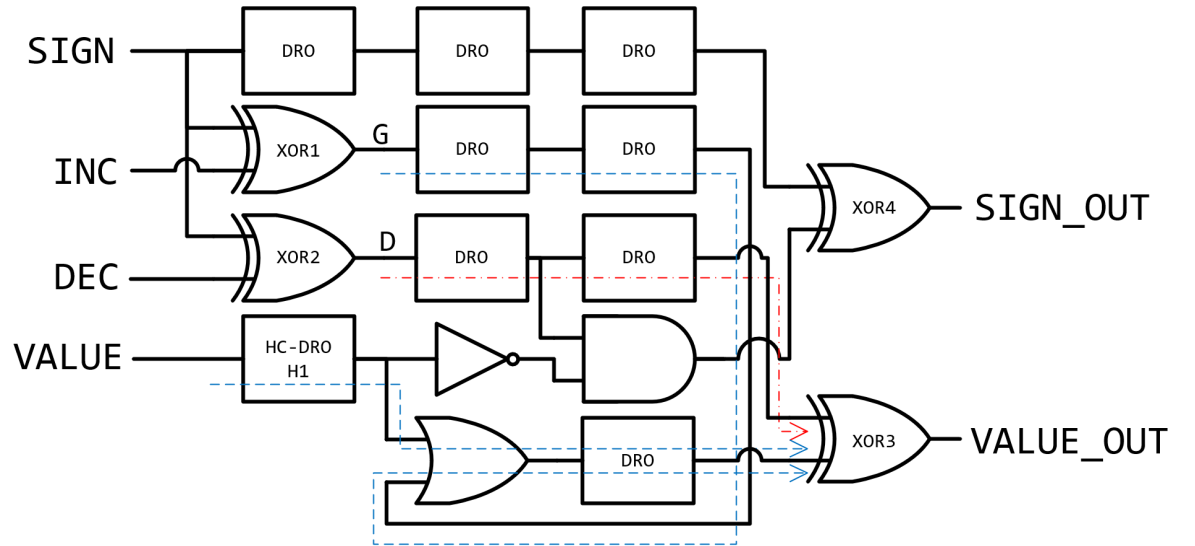
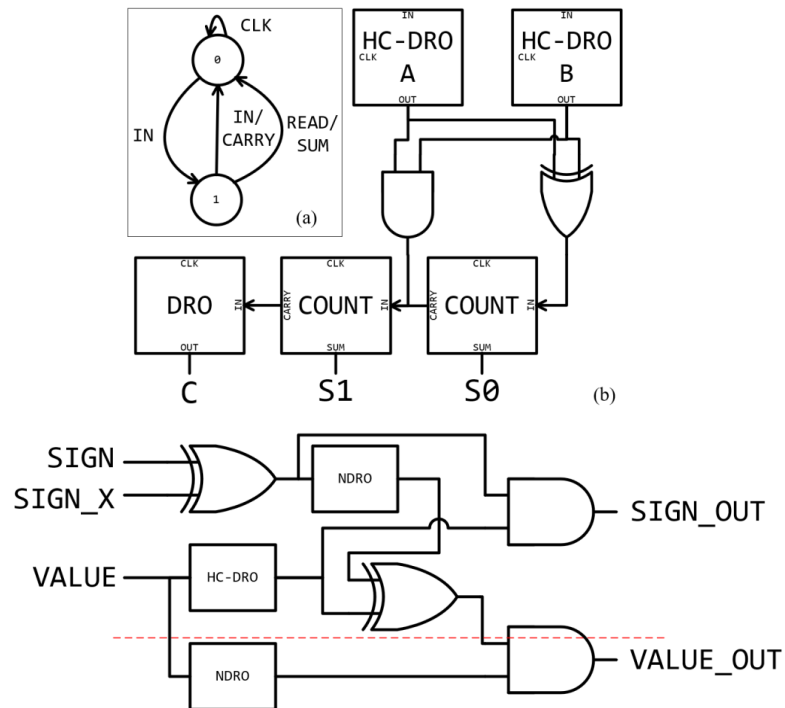
Inference and Training Unit Design

- 31 pins is a just used before shared in 13 pins easy to store



Design Without Decode/Encode

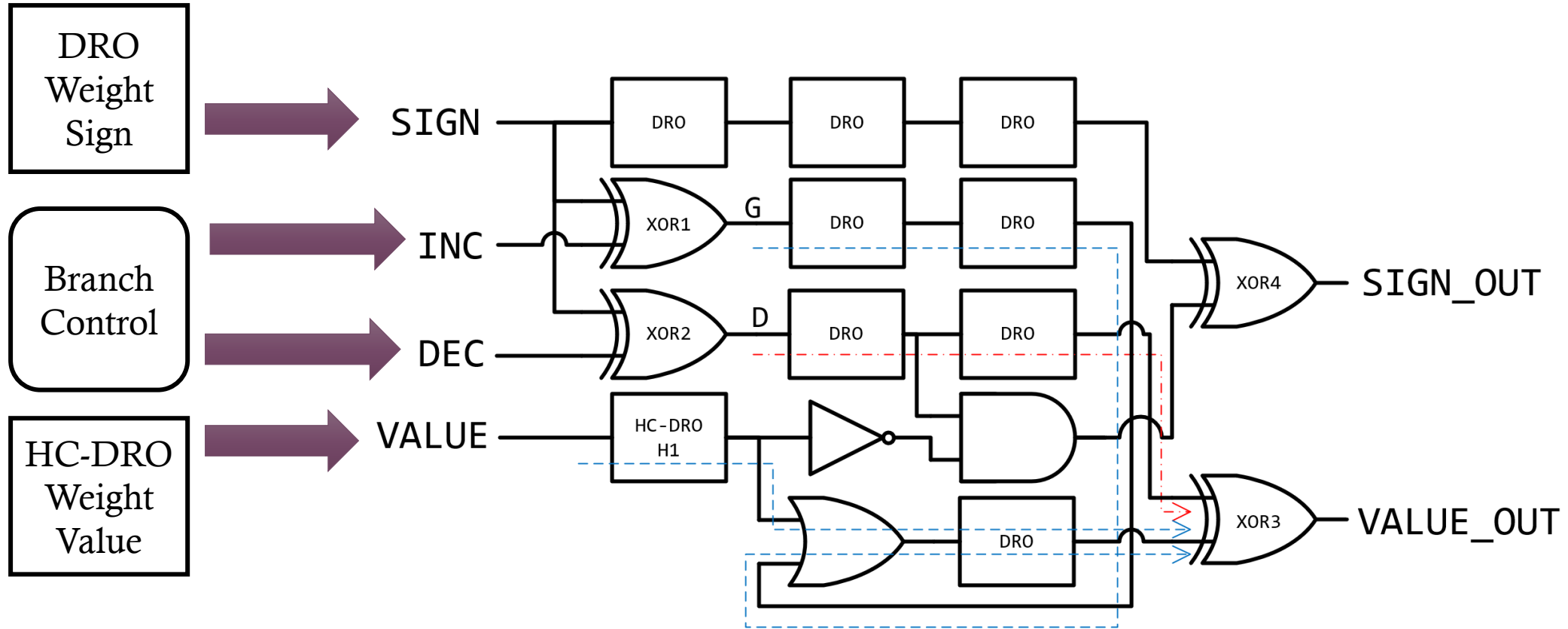
- Customized inference and training units
 - Compute directly on the un-decoded data (serial pulses)



Design Without Decode/Encode

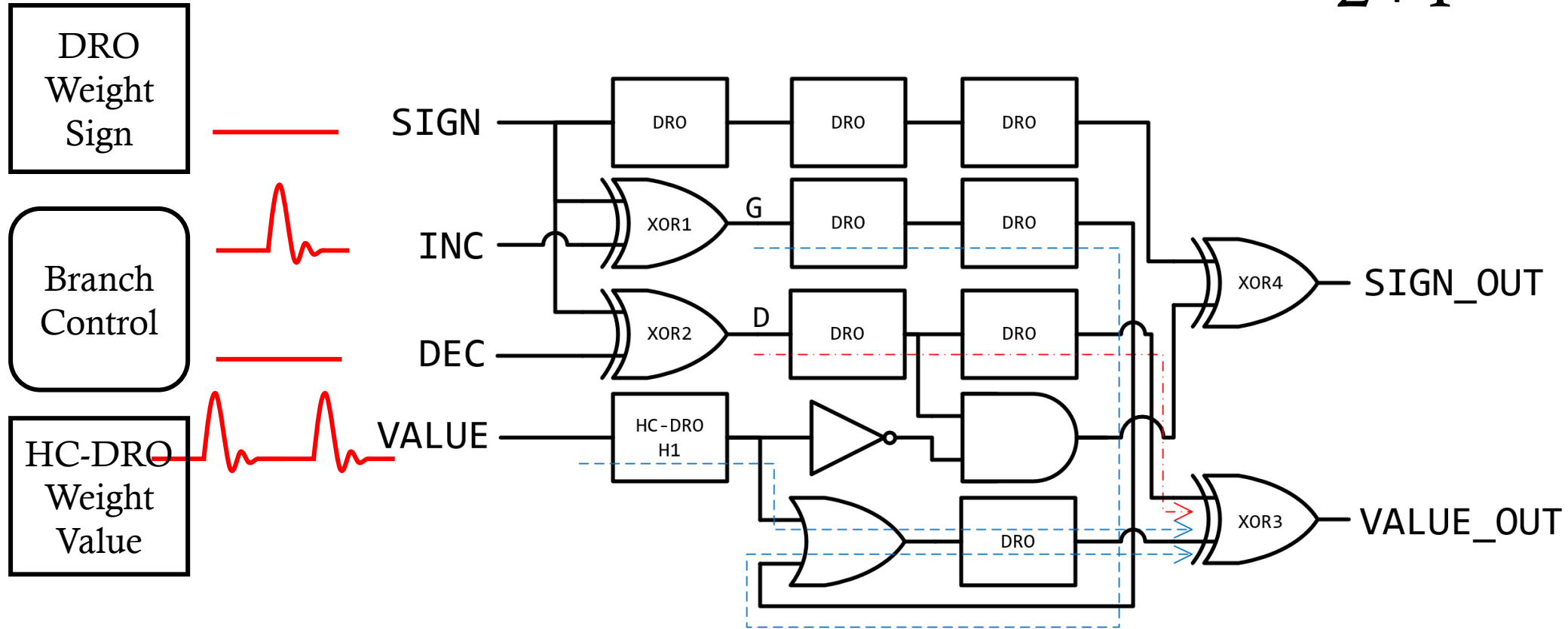
- Customized inference and training units
 - Compute directly on the un-decoded data (serial pulses)

Training Unit



Training Unit: Increment Operation

2+1



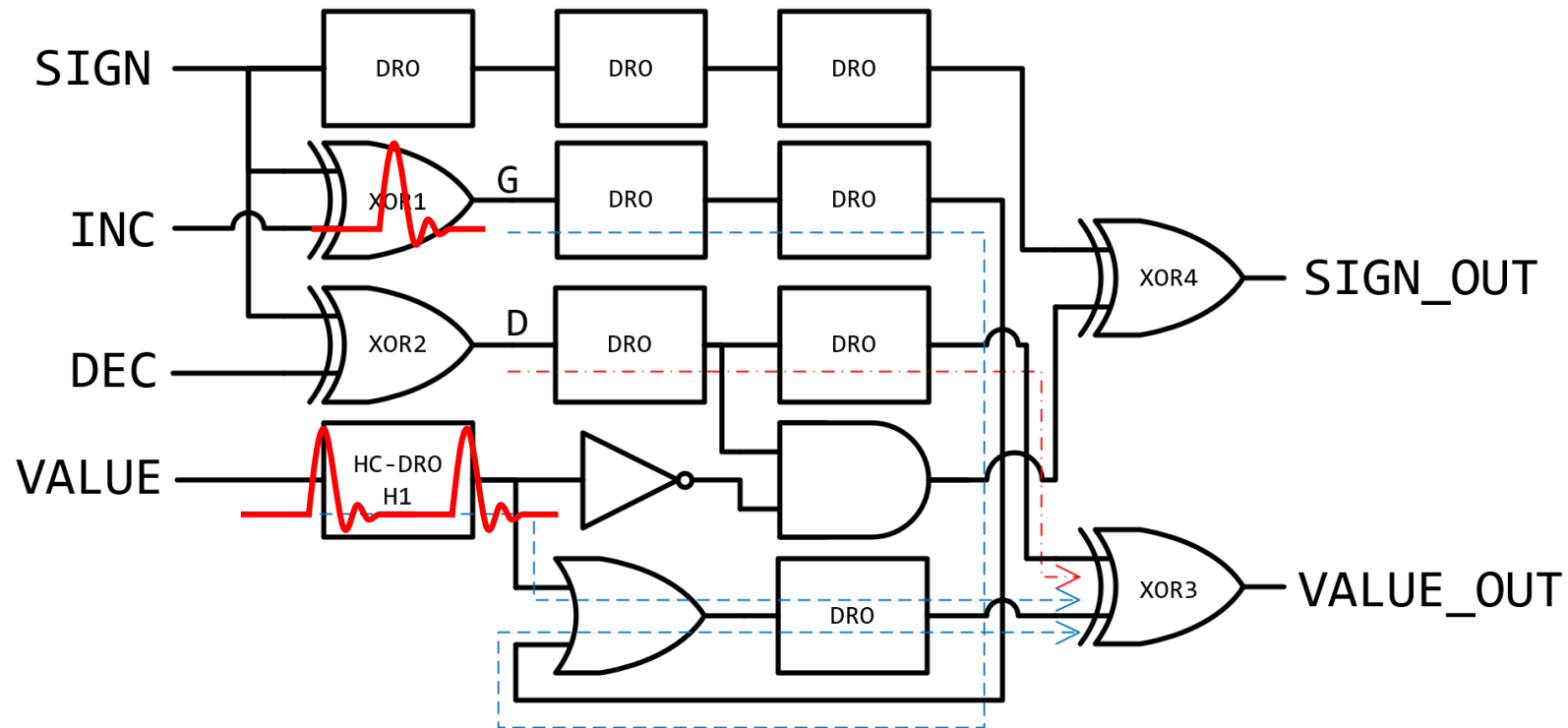
Training Unit: Increment Operation

2+1

DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



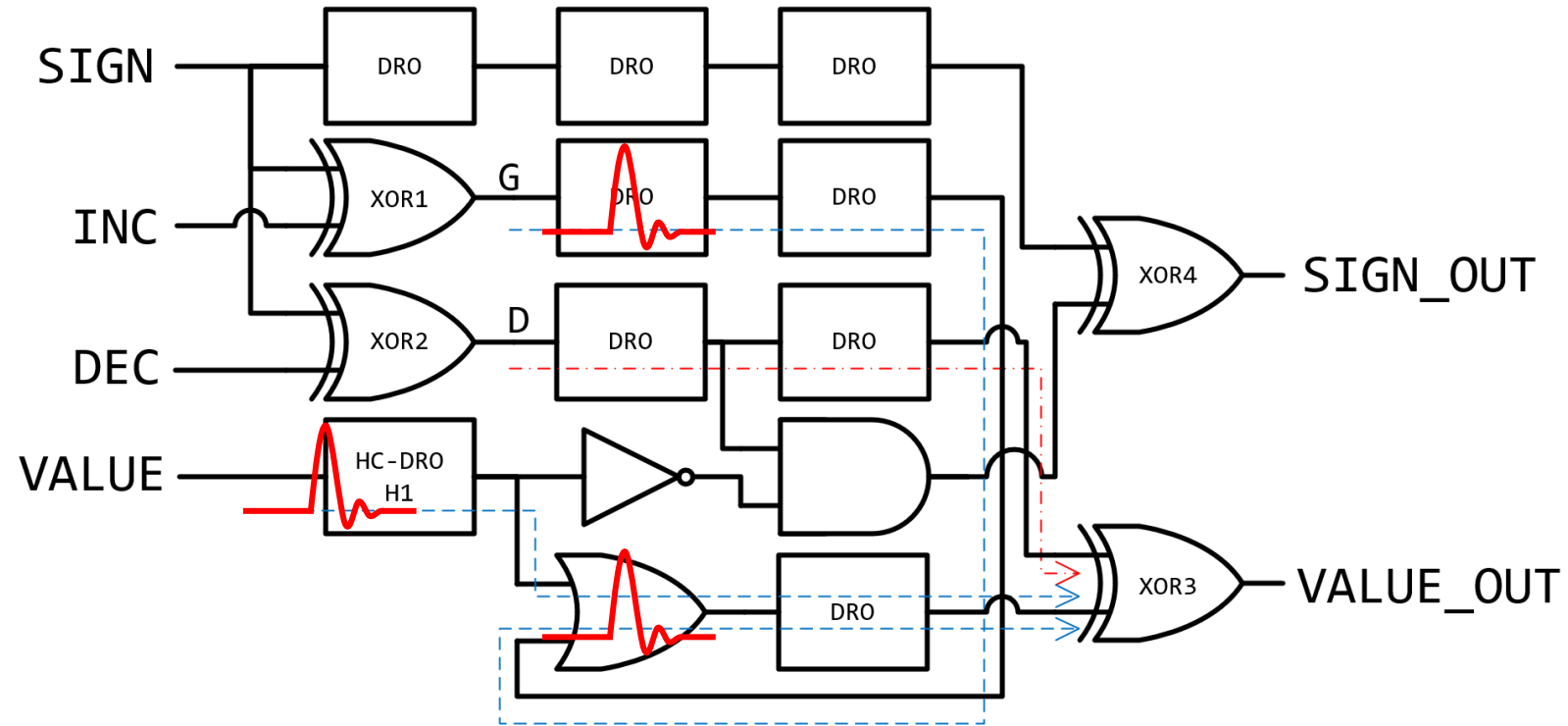
Training Unit: Increment Operation

2+1

DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



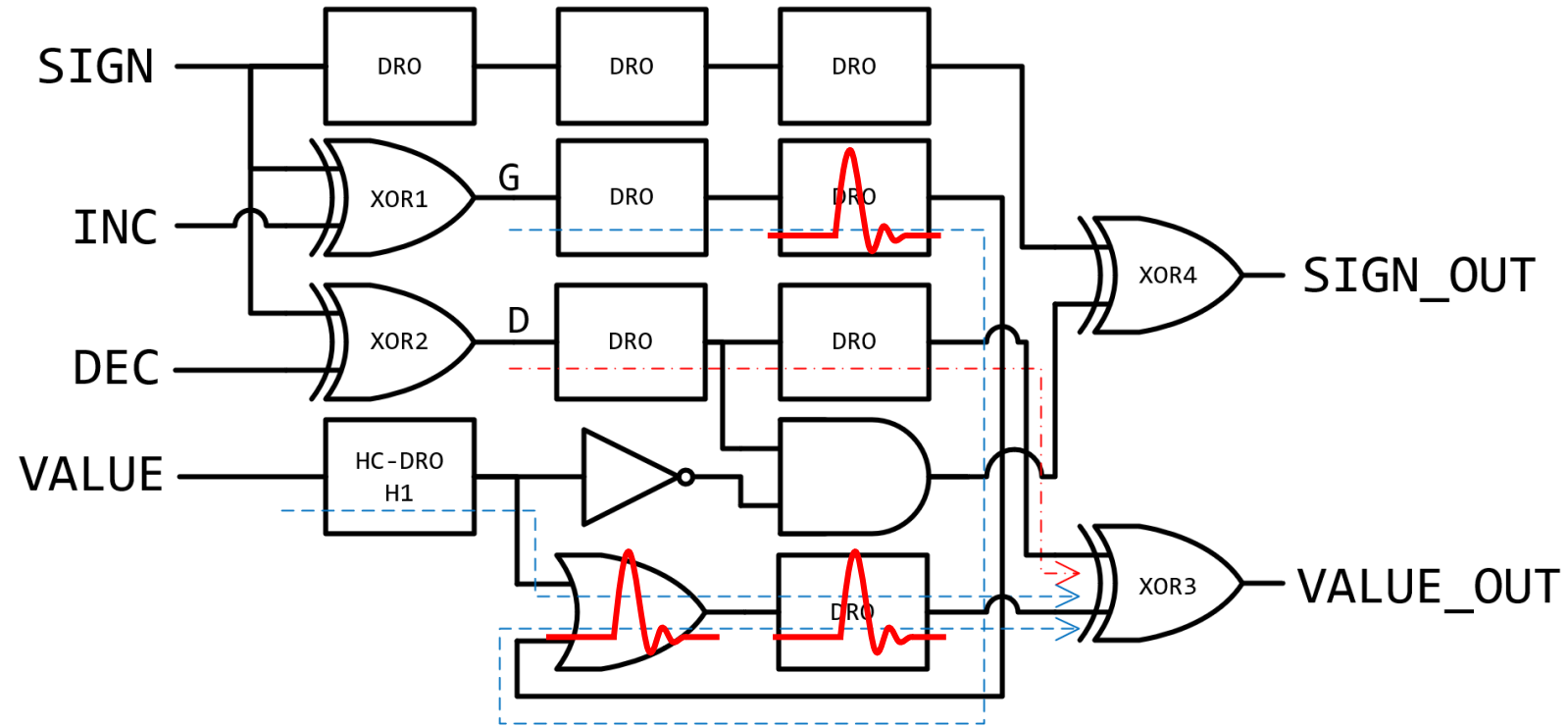
Training Unit: Increment Operation

2+1

DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



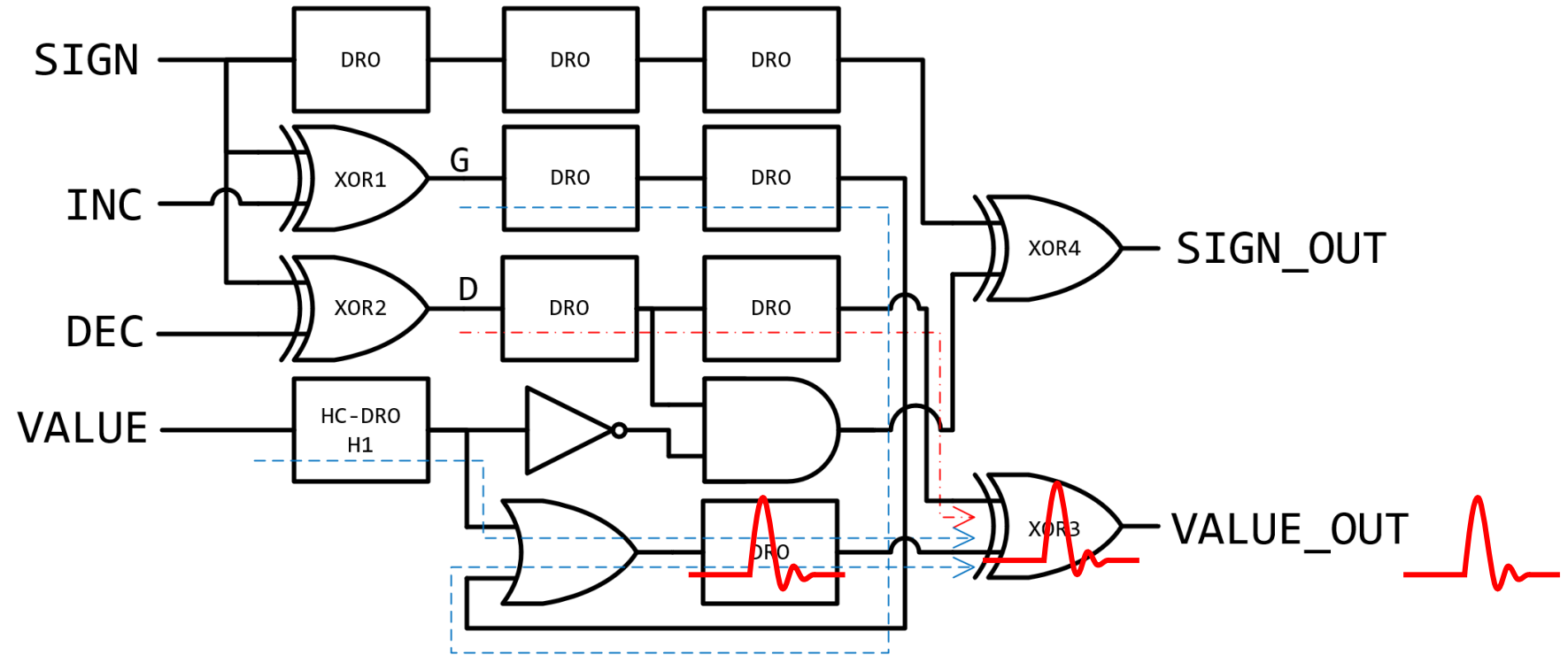
Training Unit: Increment Operation

2+1

DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



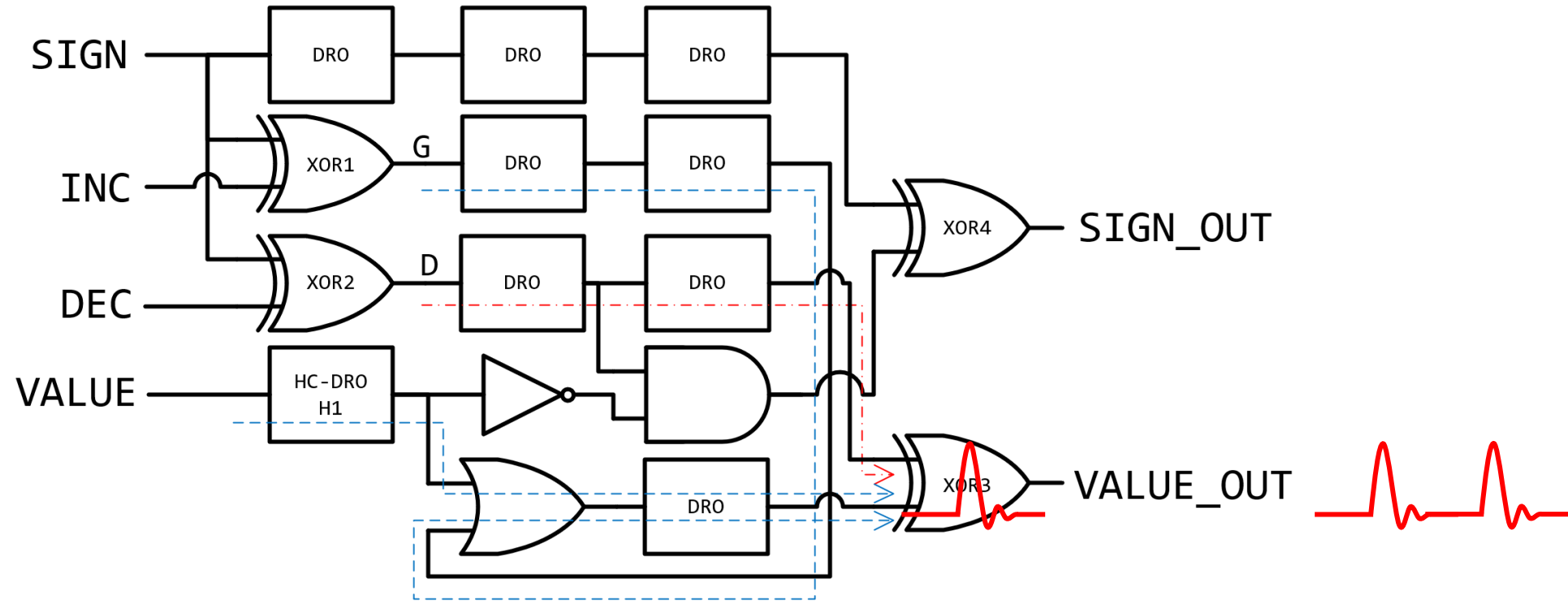
Training Unit: Increment Operation

2+1

DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



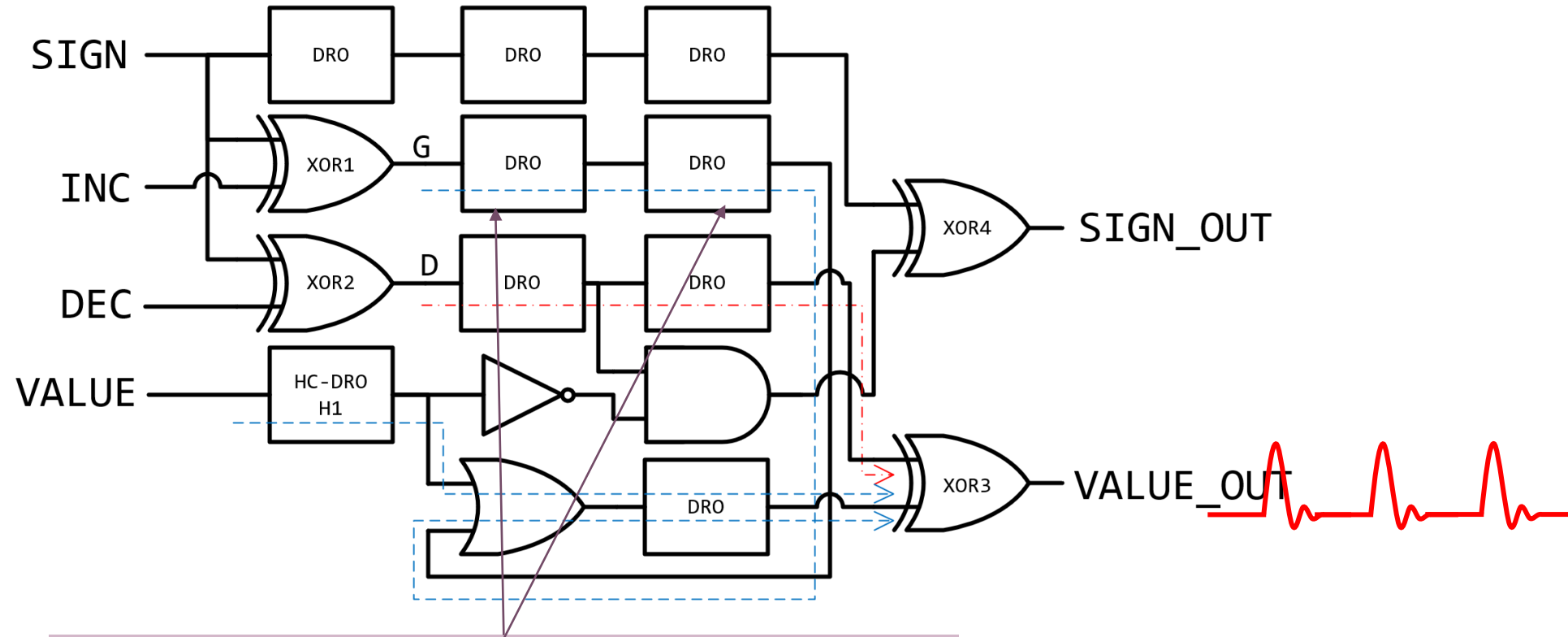
Training Unit: Increment Operation

2+1

DRO
Weight
Sign

Branch
Control

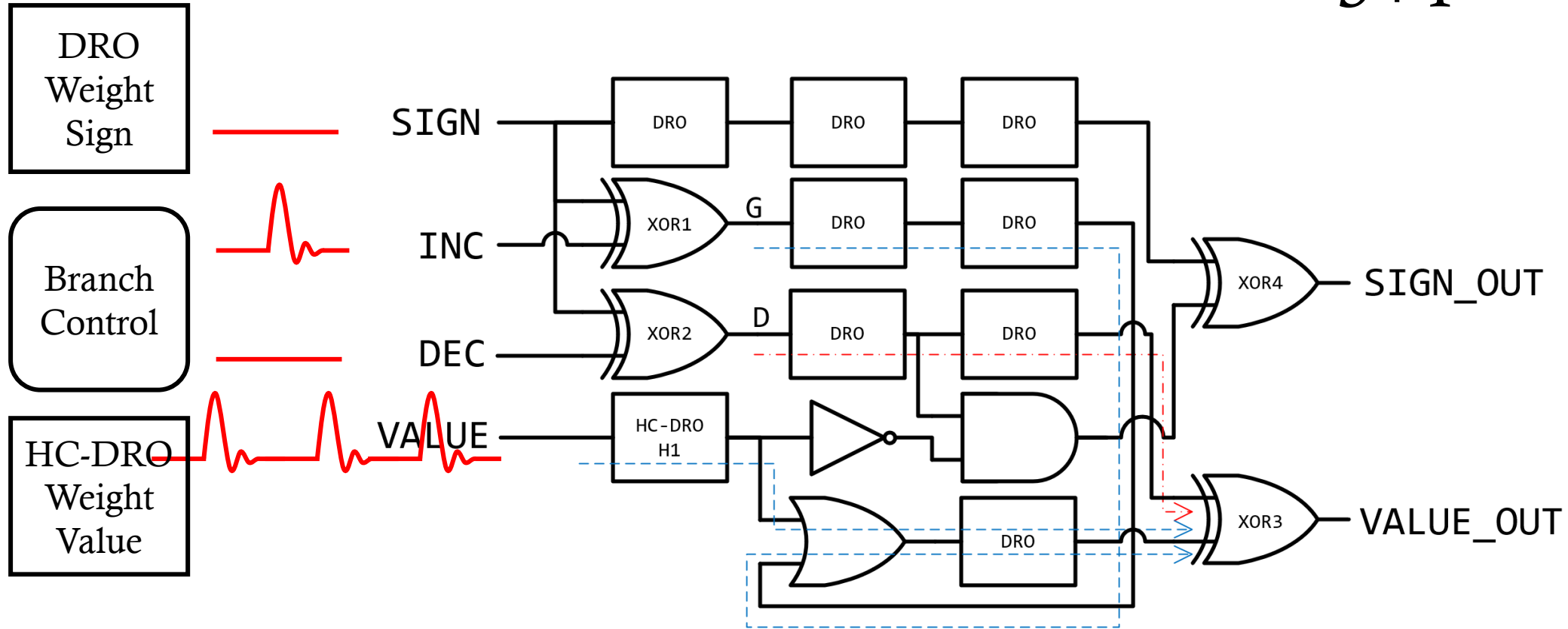
HC-DRO
Weight
Value



Addition of Two DRO Buffers to Time the Accumulation

Training Unit: Saturating Counter Design

3+1



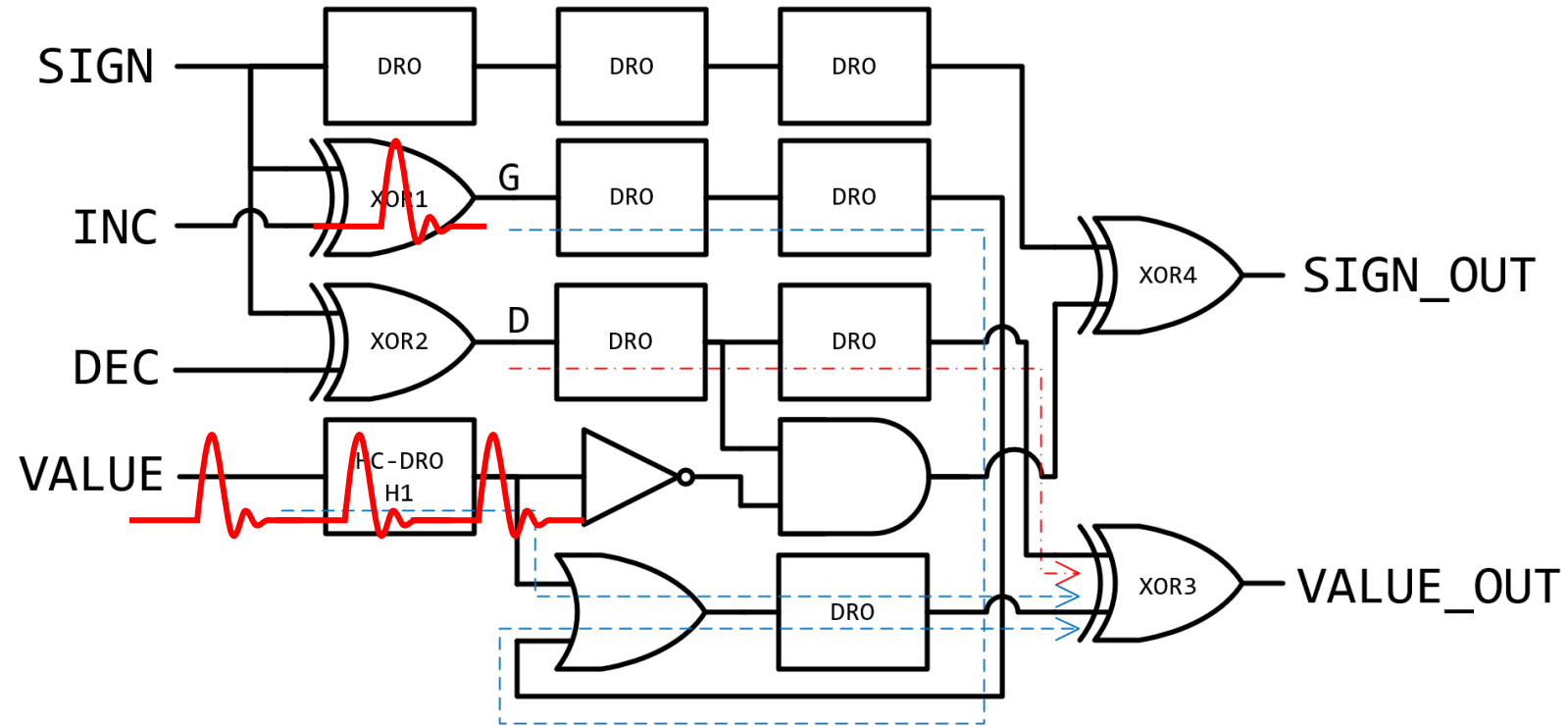
Training Unit: Saturating Counter Design

3+1

DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



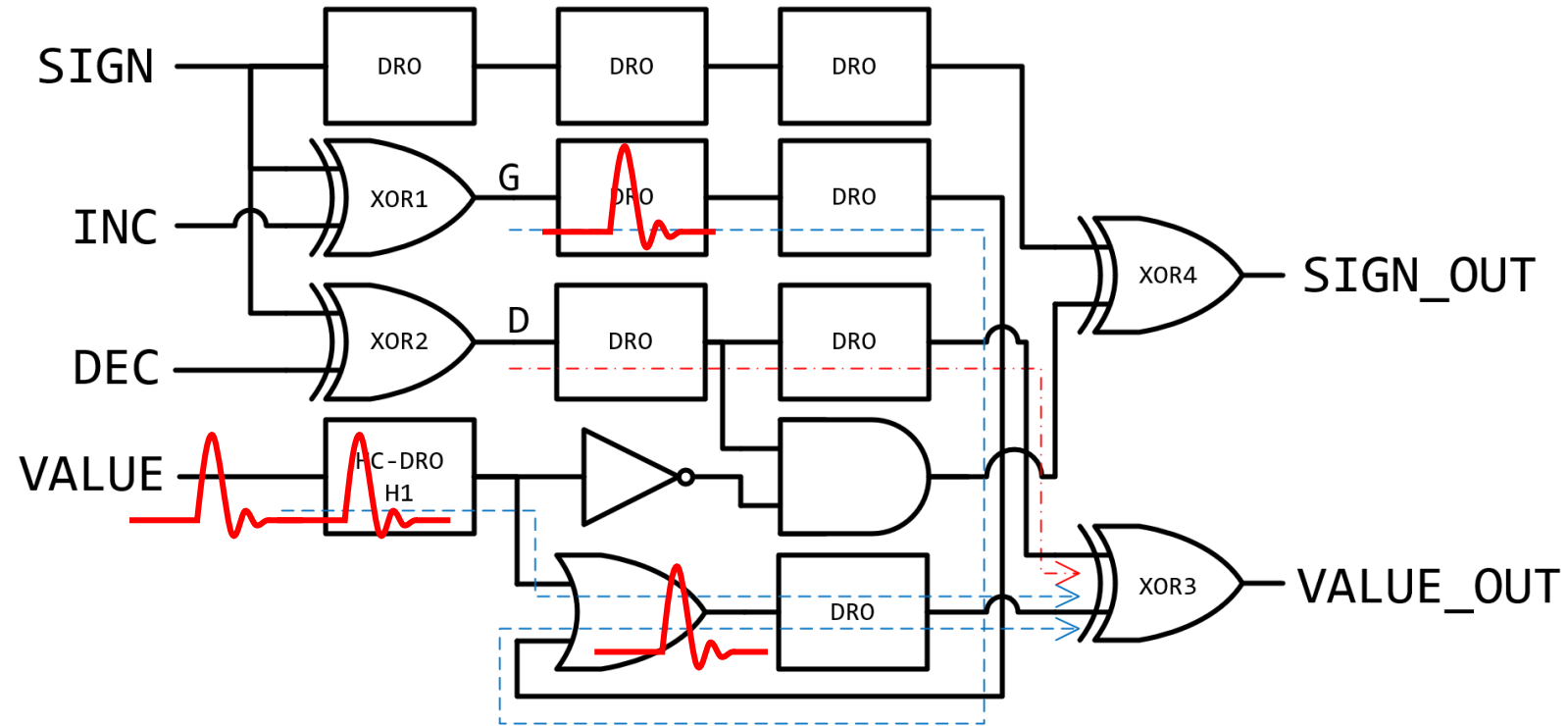
Training Unit: Saturating Counter Design

3+1

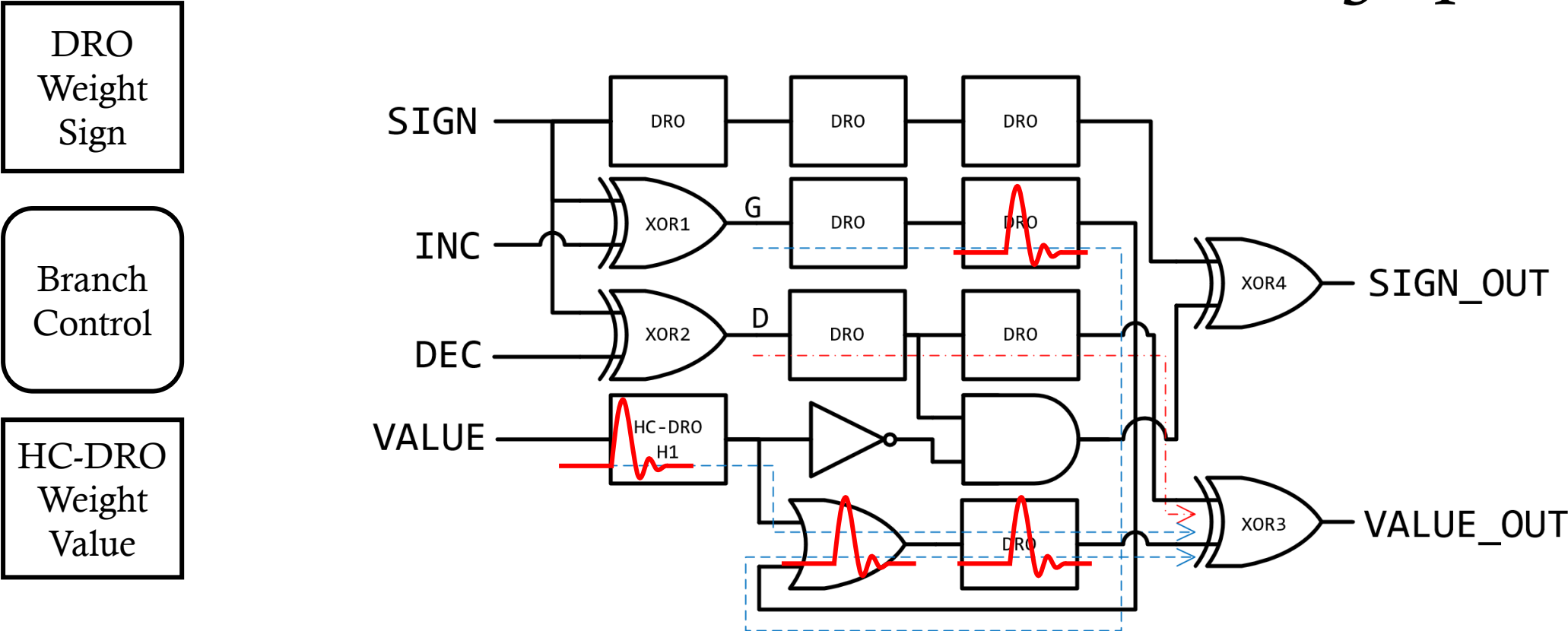
DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



3+1



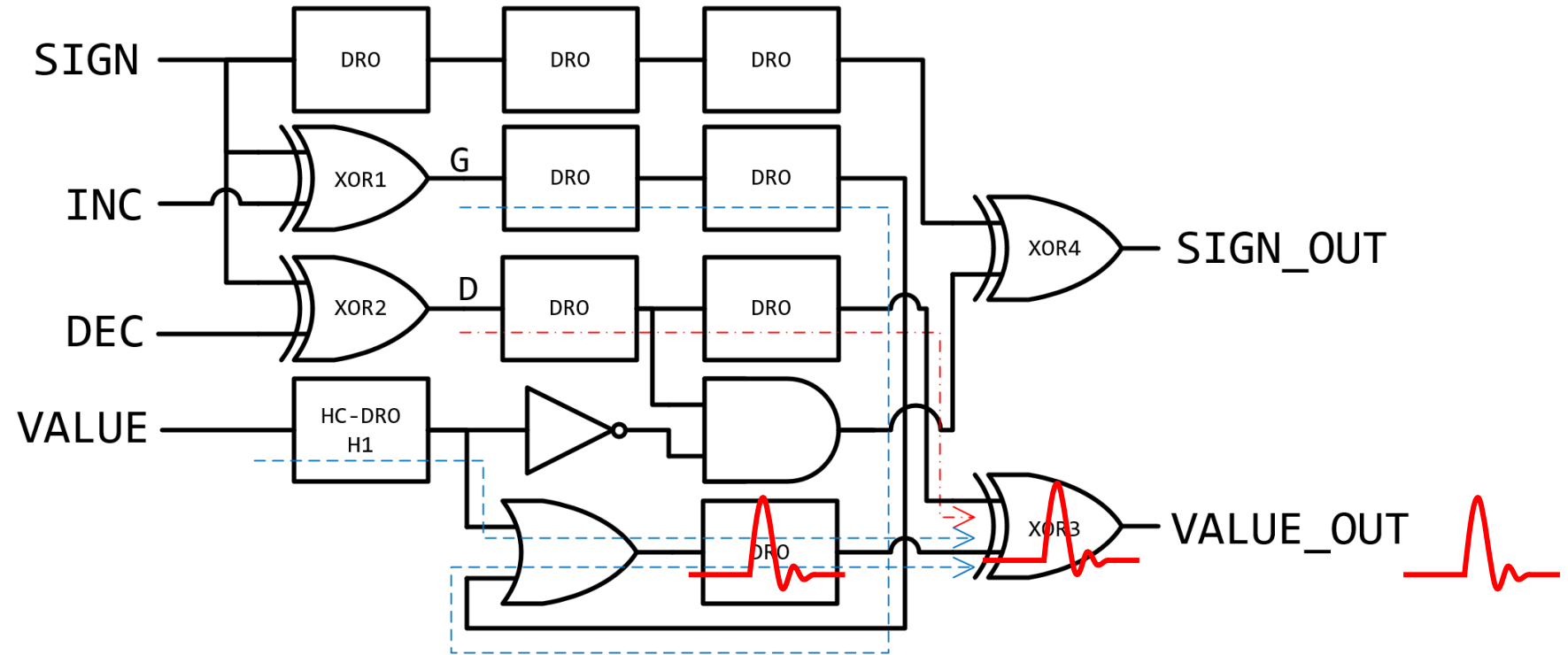
Training Unit: Saturating Counter Design

3+1

DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



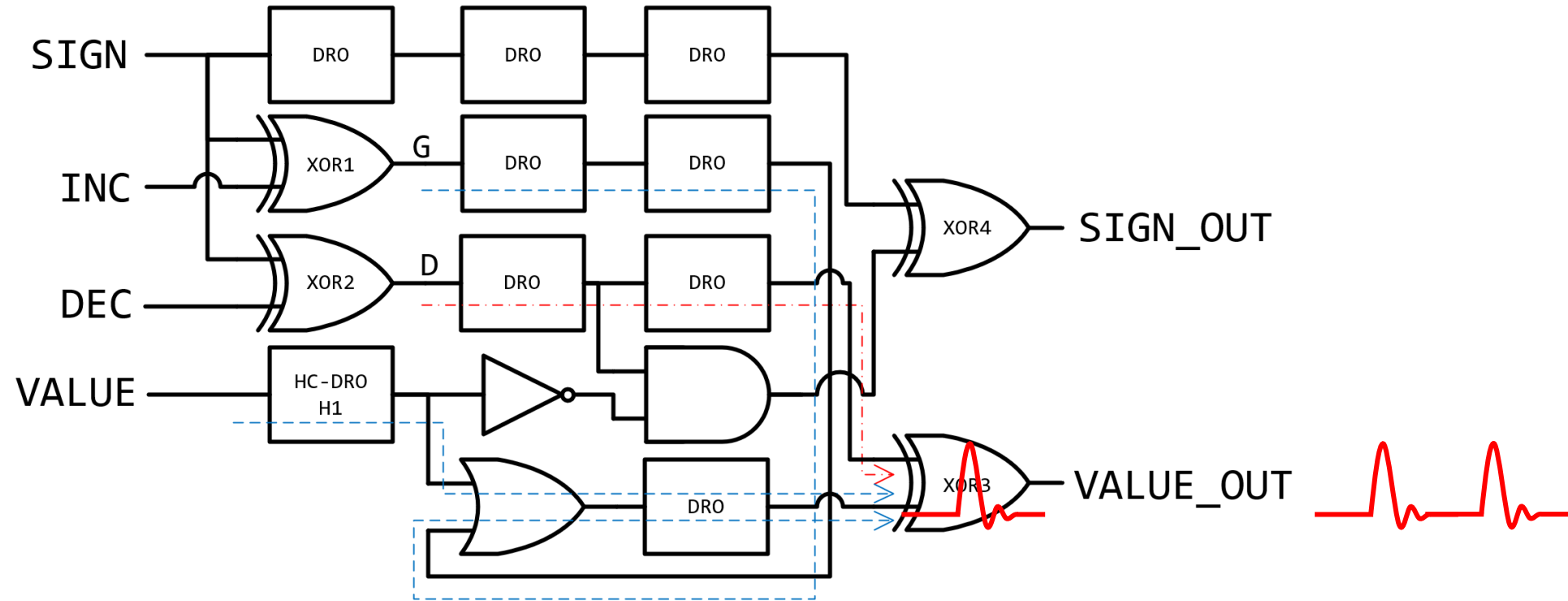
Training Unit: Saturating Counter Design

3+1

DRO
Weight
Sign

Branch
Control

HC-DRO
Weight
Value



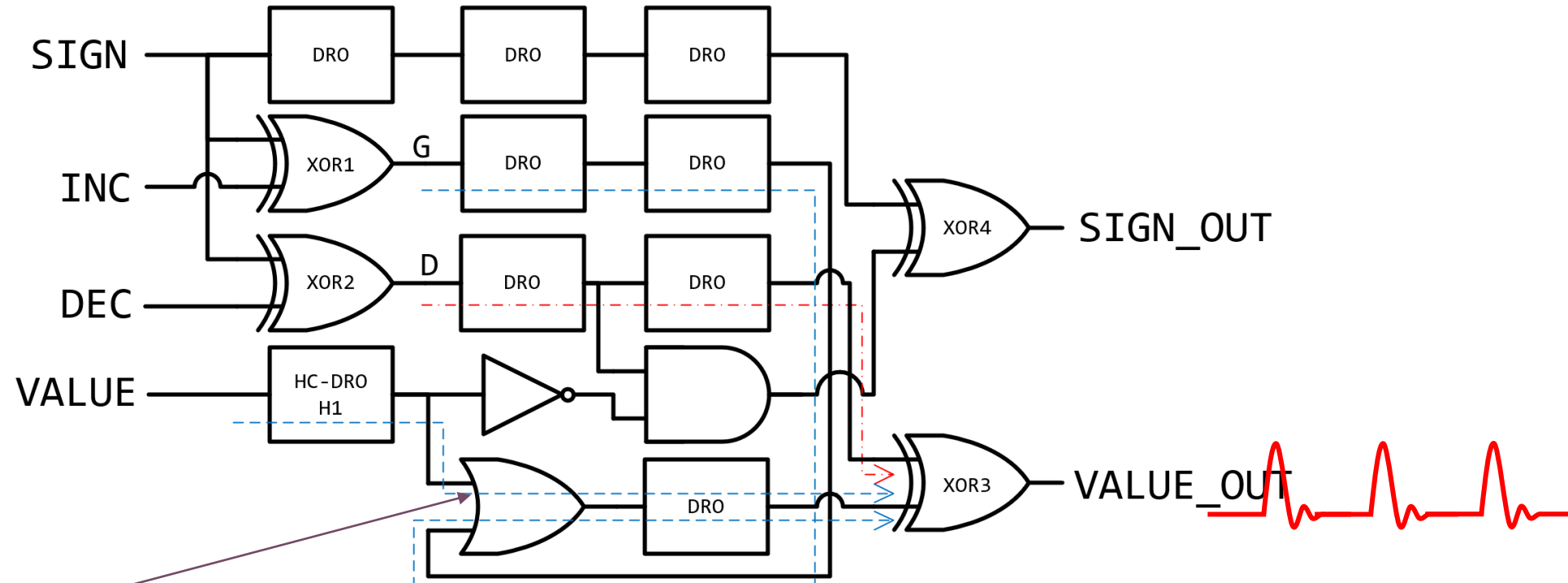
Training Unit: Saturating Counter Design

3+1

DRO
Weight
Sign

Branch
Control

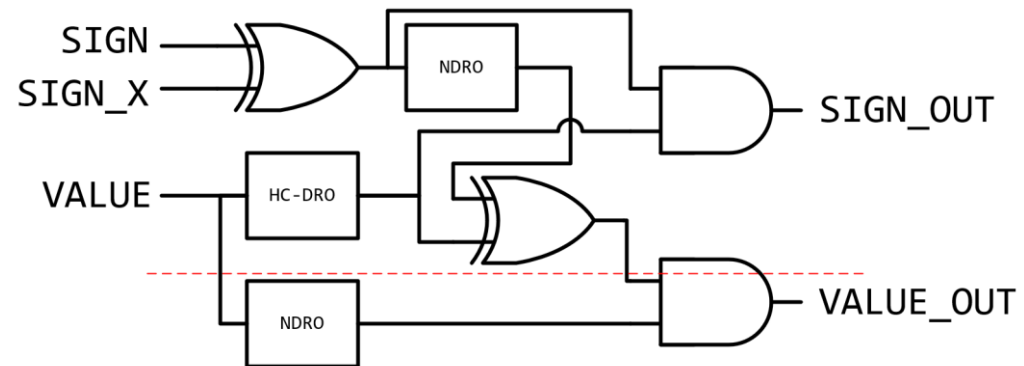
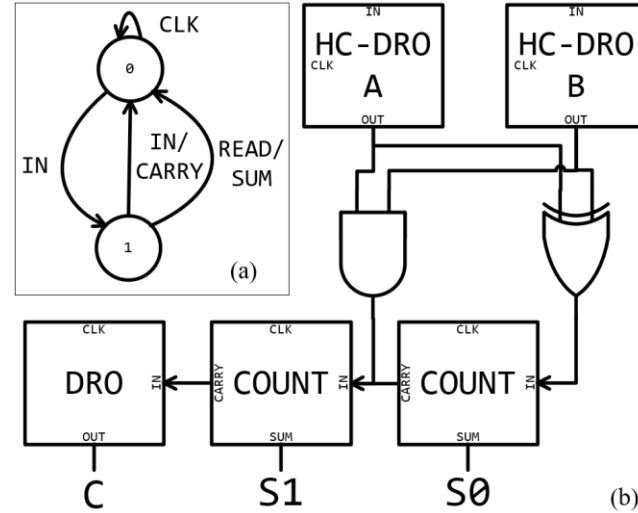
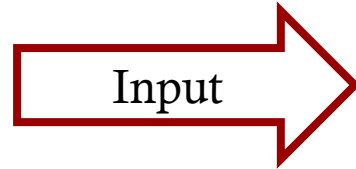
HC-DRO
Weight
Value



OR Gate strategically absorbs the increment pulse
Implements Saturating Counter

Inference Unit

- See paper for more details



Outline

- SFQ Memory Structures
- Build an SFQ perceptron branch predictor
 - Weight Storage
 - Inference and Training Unit
- **Evaluation**

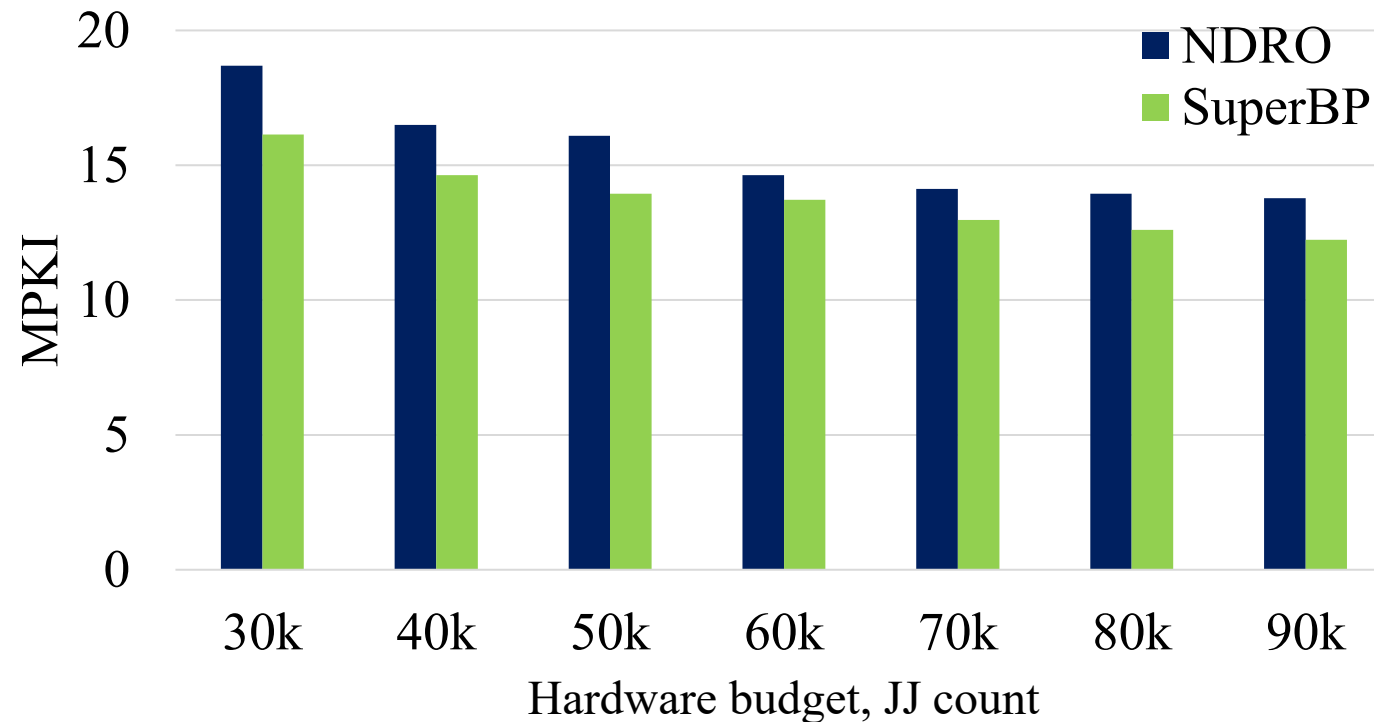
Hardware Performance

BP Size	16 x 8	32 x 16	64 x 32	128 x 32
NDRO-baseline JJ count	20516	62222	205035	369634
SuperBP JJ count	13079	39040	125784	225874
JJ Saving %	36.25%	37.26%	38.65%	38.89%

SuperBP saves ~38% JJs compared to the NDRO design

Simulation Results

- We designed an SFQ based gate-level CPU simulator. The simulator is based on the RISC-V ISA Simulator Spike and written in C++.



CONCLUDING THOUGHTS



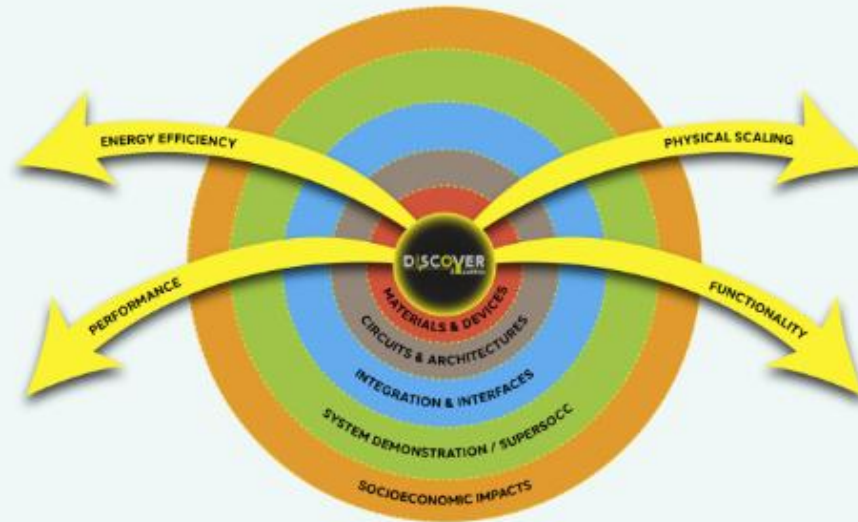
Vision

Enable scientific and technological breakthroughs

Greatly reduce energy requirements of national computing infrastructure

Develop and demonstrate superconductor electronics (SCE) and superconductive computing technology

Innovate in technology, circuit design, and architectures with targeted application



ARCHITECTURAL IMPLICATIONS



Architectural implications of SFQ technology provide exciting challenges



We are just beginning the journey



If interested, please contact us



<https://discoverexpedition.usc.edu/>