SpecQ: Speculative Local Quantum Error Correction with Superconducting Priority Selection

Abstract—As quantum computers become more fault-tolerant, the resources required for error correction continue to rapidly grow as a function of the code size and number of logical qubits in the system. In particular, the communication bandwidth between different temperatures in the quantum computer is growing at a rate that is not sustainable. We observe that we can defer and reduce communication by greedily calculating error corrections and speculatively applying them with small hardware overhead in the low-temperature layers. At the cost of accuracy and program execution time, this allows for a greater scale in terms of logical qubits and shorter quantum program execution times.

A. Background on SFQ

To minimize the communication between stages with large temperature differences, which causes issues due to heat dissipation, our hardware must be placed as close to the mK stage the qubits occupy as possible. Single Flux Quantum (SFQ) [3] is a logic family based on superconducting hardware that is also able to perform quantum gates [6], although we will only make use of it's classical aspect in this paper. In SFQ, propagation picosecond wide pulses of voltage is used to perform computation. The presence of a pulse denotes a logical 1, and the absence denotes a logical 0. The flow of pulses in a circuit is controlled by Josephson Junctions (JJ), which are the fundamental element of the technology, analogous to the role transistors play in traditional semiconductor-based logic. SFQ currently suffers from having 3 or 4 orders of magnitude lower integration density than CMOS, thus reducing size to fit in the available budget of around a million JJ per chip is the main concern when designing a circuit for SFQ. The main metric used to evaluate this is the count of JJ the circuit requires. The most widely used family of SFQ cells is RSFQ [3], which adds clock ports to many combinational gates (AND, OR, XOR). Theses cells output the result for the pulses received since the previous clock signal in their data ports when a new clock signal reaches them. The equivalent of a DFF in SFQ is the Destructive Read Out (DRO) cell, which will produce an output pulse upon receiving a clock pulse if it received one at the data port before. A variant of the DRO cell is the complementary output DRO cell (DROC), which has an additional output that triggers on the clock pulse if a data pulse has not been received. Clearly, exactly one of the DROC's output ports will fire with every clock trigger. The above cells are stateful, since they retain the information of which pulses they have received. If a DRO or DROC receive more than one data pulse before the clock trigger, additional data pulses will have no effect. Rather than having the clock signal reach all gates at the same time, a common approach is to pass clock pulses from earlier gates to later ones in a clock follows data scheme called concurrent clocking. This allows the entire circuit to be evaluated by only firing the clock trigger once. SFQ requires special interconnect cells, called splitters, to produce two pulses from one in order for a wire to have a fanout larger than one, and mergers to combine pulses from two sources in one. The need for splitters makes signals with very large fanouts, like clocks or global resets, much more expensive and hard to route for. The simplest interconnect cell is the Josephson Transmission Line (JTL), which repeats the pulses it receives. It can be used to add a set delay to a pulse's propagation. SFQ cells also include the Muller C and inverted C elements which do not require a clock port. The C element fires when pulses have arrived in both of it's two data inputs. Respectively, the inverted C element fires on the first time it receives a pulse in either of it's ports and resets when the other port receives a pulse.

I. Hardware Implementation

In order to limit the data transfer to higher temperatures, our design must operate at the 4 Kelvin layer of the cryogenic stack. Thus our design is based on the superconducting SFQ technology, which is the most mature computing technology for operating at this temperature. As previously discussed, the integration density of SFQ circuits is far behind that of modern semiconductors and the rate at which dissipated heat can be cooled in this environment is limited, limiting us to a few million JJ with a power budget of 1-2 Watt.

There are three functionalities to implement in the 4K stage: a) selecting the logical qubits with the highest priority scores, b) producing the priority score for each logical qubit, and c) locating the data qubits that should receive speculative correction.

A. Arbiter implementation

In order to use the priority scores determined by our scheme to allocate the data transfer budget to the highest scoring logical qubits, we need a way to identify the k logical qubits with the highest priority scores out of the n that are competing to be decoded. An arbiter circuit that implements the top-k argmax of the n priority scores based on arithmetic comparison of binary priority scores and multiplexing of qubit indexes would incur a significant area cost for the budget current integration density of superconducting cells allows, especially as the number of logical qubits n and number of supported priority levels increases. Fortunately, designs like this one that consist of comparisons can be implemented efficiently using temporal computing, a logic scheme where information is represented as the time of arrival of signals, allowing priority scores of any number of bits to be supported by a single wire. Specifically, we design our arbiter using the paradigm of race logic [4], for which a mapping of it's primitives to superconducting cells was provided in [7].

For the comparator module we base the arbiter on, we make use of three race logic primitives. The first two primitives are First Arrival (FA) and Last Arrival (LA), which perform the equivalent of min and max operations of the temporally encoded values. FA will fire upon the first pulse it receives in one of it's input ports and reset upon receiving a second pulse on it's other input port, whereas LA both fires and resets upon receiving the second pulse. In SFQ they are implemented with an inverted Muller C cell and a regular C cell respectively. We have tuned these cells to have very similar delays, so differences in their propagation speed do not lead to the need for synchronizing signals inside the arbiter. The third primitive is Inhibit (INH), that has two input ports a and b and propagates a pulse received on b only if it arrives before a pulse on a does. Typically Inhibit is implemented using a clocked inverter cell, however because we need only a boolean signal of whether b arrived before a and not a temporal signal of when b arrived for our purposes, we substitute it for the cheaper DRO cell, and assign a to the DRO's clock port and b to it's data input port. In the case a and b arrive very close to each other, they will both represent the same priority score and their comparison will be a tie, in which case either the DRO firing or not are both valid for our purposes. With these primitives we can design the comparator module of size 2 that will be the building block of our arbiter.

The comparator module's ports consist of four pairs, each pair consisting of a data port and a select port. It provides the functionality for both a forward pass that uses the data ports to sort the input priority scores and a backward pass that uses the select ports to route signals to go down the same path data signals did in the forward pass. The forward pass connects the x_0 and x_1 input wires which encode priority scores to FA, LA and INH cells. The outputs of FA and LA connect to the x_{min} and x_{min} output ports of the comparator, propagating the earliest and latest pulses from the inputs respectively. The output of the INH cell goes to the data input of two DROC cells, to prepare them for the backward pass. In the backward pass input ports sel_{min} and sel_{max} are connected to the clock ports of the DROC cells. The outputs of the DROC pair are merged such that a pulse from sel_{min} is routed to sel_0 if the DROC pair stores a pulse from the INH, meaning that signal x_0 arrived before x_1, and to sel_1 otherwise. Similarly, a signal from sel_{max} goes to sel_1 if x_0 arrived before x_1 and to sel_0 otherwise.

Fig1(b) shows an example of the comparator module's operation with the routes the various signals took. Fig 1(a) shows the gate level implementation of the comparator module. As it can be seen, only a handful of gates are needed compared to what a bit parallel arithmetic implementation would require.

To build the full arbiter that provides the functionality of top-k argmax, we arrange comparator modules in the configuration of a bitonic sorter. Fig 1(c) shows the arrangement for a 4 element bitonic sorter. Also shown is the path of the latest arriving encoded priority score through the sorter and the path a select signal given at the highest position of the sorter's final layer takes through the comparator modules to fire at the select port of the first layer, denoting the argmax of the data input pulses.

Scaling the bitonic sorter configuration to the number n of priority scores we need to evaluate would work for a top-k argmax if we first send the temporally encoded priority scores for all n qubits as input pulses for the forward pass and after that fire pulses at the select ports of the final layer corresponding to the k largest positions. These would get routed to the select ports of the first layer corresponding to the k largest scores entered in the data ports, providing a binary flag representation of the argmax that can be used without additional decoding. However, a large part of this structure would be wasted on comparators sorting signals that cannot be in the top-k, and thus are not needed. After removing the unnecessary comparators we are left with a structure where the data signals are split in groups of length k and each group goes through a full bitonic sorter of size k. The number of data signals is then cut in half by selecting the k largest outputs from each pair of k-sized sorters. For every comparator that performs this reduction only the max pair of data/select ports is connected to the next layer, whereas the x_{min} and sel_{min} are left unconnected. The $\frac{n}{2}$ signals left go through a similar sort and reduce process, although subsequent blocks of size k sorters require less comparators. After the final reduction k signals will be left, from which the select signals for the top-k will originate. Fig 1(d) shows such a sparse arbiter that selects the top 4 out of 16 signals.

For a constant ratio of qubits to be selected to total qubits $\frac{k}{n}$, the arbiter scales at a rate of $O(\log^2(n))$ latency-wise and $O(n\log^2(n))$ area-wise to the number of input logical qubits. Because the area scales superlinearly, for large numbers of logical qubits it can be more efficient to partition them into subsets and dedicating a smaller arbiter to each subset. Each of these arbiters will be assigned a slice of the total available bandwidth to allocate to the qubits it manages.

B. Reset scheme

After an execution of the arbiter, some of the DRO and DROC cells could be still storing a pulse, which needs to be cleared before the next execution can occur. The FA and LA cells have all received a pulse in both of their input ports during execution, thus they are already reset. As the arbiter only operates once per decode cycle, hundreds of nanoseconds pass between executions, providing ample time to reset the DRO and DROC cells. Although an external reset signal could be used, it would require more complicated and expensive cells that have an additional reset port and incur splitting overhead to connect the reset signal to every cell. To avoid this, we design a data-driven reset scheme. This enables us to return all cells to their original state by only inserting signals to already existing ports in the design, with no changes to it's internals. The first step of the reset scheme resets all the inhibit DRO cells via a forward pass of the arbiter. When a comparator module gets a pulse at x_0 first and at x_1 after, it's inhibit DRO will fire after receiving a pulse at the data port and thus clear. Additionally in this case the comparator will propagate the data inputs with the same order it received them.

We make use of these properties by inserting a linearly increasing sequence of pulses in the data input ports of the arbiter, meaning that input $i+1$ will receive a pulse a fixed time after input i. Due to the arrangement of comparators in a bitonic sorter, the order of pulses will remain the same through every comparator layer, as it is already sorted, and also every comparator will receive a pulse at input x_0 before x_1, clearing it's DRO. Since every inhibit DRO in the design fired, all DROC cells in the arbiter's comparators will store a pulse, setting the stage for the second step of the reset scheme. In this step, we insert a pulse at the select signals for the top k scores in the final layer, as well as the unused sel_{min} ports of the comparators in every reduction layer. Together these n select signals will pass through every select port in their path towards the first layer, causing every DROC to fire and clear in the process. The linearly increasing sequence of pulses for the first step can be easily produced using a splitter chain, where a pulse entering the i-th chain link will be split into two pulses, one being merged into the i-th input of the arbiter and going through a few JTL to add a bit of delay before entering the $i+1$ part of the splitter chain. To support the second step only a splitter tree of size n for the reset signal is needed.

C. Priority score evaluation

To evaluate the priority score of a logical qubit from it's syndromes, the first step is producing the complex flags for each ancilla qubit. We use the same basic gate configuration as [5] for this end, which consists of only three layers of gates (XOR, XNOR and AND). Since the circuit is not pipelined but instead gets run once all the way through every decode cycle, we route in a clock-follows-data fashion (concurrent clocking), which lets us avoid inserting buffer gates to deal with logic depth imbalance. Beyond their role in determining priority scores, these complex flags are used to select the data qubits that will have an X or Z flip applied to them in the case of speculative correction. This speculative application circuit is entirely separate for each data qubit and consists of 4 splitters, 2 AND gates and one XOR gate, at a cost of 54 JJ per data qubit.

The complex flags from each ancilla qubit of the same type are aggregated into a single flag for each quadrant that tracks whether any qubit belonging to the quadrant is the center of the complex clique. This aggregation is done very efficiently by connecting the complex flag wires of each quadrant's ancillas as inputs to a merger tree of the same size. Fig 2(a) shows the connections of complex flags to the merger trees for an example surface code. Since merger cells are both clock-less and stateless, this is the cleanest way to combine a set of signals. Because more than one pulse might reach the output of the merger tree if multiple input complex flags fire, a DRO is placed at the output of each tree. This way any number of pulses at the output simply set the state of the DRO to one, making it fire a single pulse when the clock triggers it.

This produces complex flags $C_i, i \in [0,3]$ for each of the four quadrants, which in combination with the flag T for whether the qubit is before a T gate will be encoded into a priority score using the system described above. The priority score mapping used for this example is shown in Table ??. To start, the four binary complex flags are sorted so all the 1s are at the beginning, resulting in signals $S_i, i \in [0,3]$ for which $S_i \geq S_{i+1}$. This sort uses a size 4 bitonic sorter similar to the one shown for the arbiter, but since the select part is not needed it becomes much cheaper, needing only 12 FA or LA cells and 12 splitters for the whole sorter. Since this small sorter is only active for a few picoseconds in a period of hundreds of nanoseconds, a small internal leakage can be used to reset the FA and LA cells after operation. In order to encode the priority score into a temporal signal for the arbiter, we pass the S flags and a dual rail representation of the T flag into a controllable delay module.

The basic block of the delay module is shown in Figure 2(b). A control signal determines which of two paths with different delays a pulse arriving at the clock port of the DROC will take before being merged to the output of the block. The additional delay incured when the control signal is set is configured by the number of JTL cells added to the longer path. Using the S flags as the control inputs, the temporal encoder consists of two chains of delay blocks. We set the JTL delay of block $B_{i,t}, i \in [0,3], t \in [0,1]$ such that the sum of the JTL delays for it and the blocks before it sum up to the temporal encoding for number of active quadrants equal to $i+1$ and $T = t$. The circuit diagram for the temporal encoder and the additional delay of each block to encode the mapping of Table ?? is shown in Figure 2(c). The two chains produce two temporal encodings, one corresponding to T being active and one for T inactive. The outputs of the chains connect to the clock port of a DRO each, the first DRO having it's data port connected to the positive wire of the dual rail T flag and the other to it's negative wire. Thus the temporal encoding is multiplexed by the T flag to enter the arbiter.

The JJ cost of the part of the circuit that produces complex flags scales linearly to the number of ancilla qubits, or quadratically to the code distance. The priority score encoder remains the same for any surface code size except for the merger tree, which scales as little as possible at one additional merger per ancilla qubit. Regarding latency, doubling the number of ancilla qubits only increases delay by one splitter in the complex flag clock tree and one merger in the merger tree. This means surface codes of very large sizes can be handled with negligible additional delay.

D. Hardware eval

In order to verify the correctness of our hardware designs, we have created gate-level implementations of each part (complex flag generation, priority score encoding, temporal arbiter). These implementations were then verified using the pulse transfer simulator Pylse [1] using extensive model-based testing. To perform the verification, thousands of input testcases were generated by sampling the distributions we observed during training. The measured outputs of our circuit simulation for these cases were compared to the results predicted by higher order models. The majority of the gate models used in the implementations are based on the open source RSFQ cell library from Sun Magnetics [2], supplemented with cell models we implemented and evaluated using WRSpice [8], a SPICE program that supports Josephson Junctions, like the Muller C and inverted C elements that were tuned to have almost identical delays. In Figure 3 we show the plot of a simulation showing the pulses in the data input (x) and select output (sel) wires of a small arbiter. The arbiter selects the two largest priority scores out of four ($k = 2, n = 4$). A small number of operands was used for this simulation in order to keep the plot interpretable. These simulations additionally provide accurate numbers for the latency and JJ count of various circuit components. These are given in Table I and Table II. As shown, the area cost for generating priority scores is fairly low for each logical qubit at only a few thousand JJs, and the overhead of the arbiter stays at the order of tens of thousands when managing many logical qubits ($n = 256$). This hardware cost is low enough to make our design viable for today's limited integration density, which is the largest roadblock for hardware meant to operate at the 4 Kelvin environment. Additionally, the combined delay of both priority score generation and arbiter barely crosses the nanosecond mark for the largest sizes considered, which is negligible compared to the total decoding time.

TABLE I
Area and delay measurements for arbiters supporting various numbers of logical qubits.

k	n	JJ count	latency (ps)
2	64	6971	379
4	128	25793	588
8	256	83345	847

TABLE II
Area and delay measurements for complex flag generation (CPX) and encoding to priority score (ENC).

d	CPX JJ cost	ENC JJ cost	CPX latency	ENC latency
3	176	390	95	143
5	592	430	105	157
7	1244	490	109	164
9	2132	570	116	171
11	3256	670	116	171
13	4616	790	119	174
15	6212	930	119	174
17	8044	1090	126	181
19	10112	1270	126	181
21	12416	1470	126	181

References

[1] M. Christensen, G. Tzimpragos, H. Kringen, J. Volk, T. Sherwood, and B. Hardekopf, "Pylse: A pulse-transfer level language for superconductor electronics," in Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2022, pp. 671–686.

[2] C. J. Fourie, K. Jackman, J. Delport, L. Schindler, T. Hall, P. Febvre, L. Iwanikow, O. Chen, C. L. Ayala, N. Yoshikawa et al., "Results from the coldflux superconductor integrated circuit design tool project," IEEE Transactions on Applied Superconductivity, 2023.

[3] 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.

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

[5] G. S. Ravi, J. M. Baker, A. Fayyazi, S. F. Lin, A. Javadi-Abhari, M. Pedram, and F. T. Chong, "Better than worst-case decoding for quantum error correction," in Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. ACM, 2022.

[6] V. K. Semenov and D. V. Averin, "Sfq control circuits for josephson junction qubits," IEEE transactions on applied superconductivity, vol. 13, no. 2, pp. 960–965, 2003.

[7] G. Tzimpragos, J. Volk, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Shalf, and T. Sherwood, "Temporal computing with superconductors," IEEE Micro, vol. 41, no. 3, pp. 71–79, 2021.

[8] Whiteley Research Incorporated, "Wrspice," 2022. [Online]. Available: wrcad.com

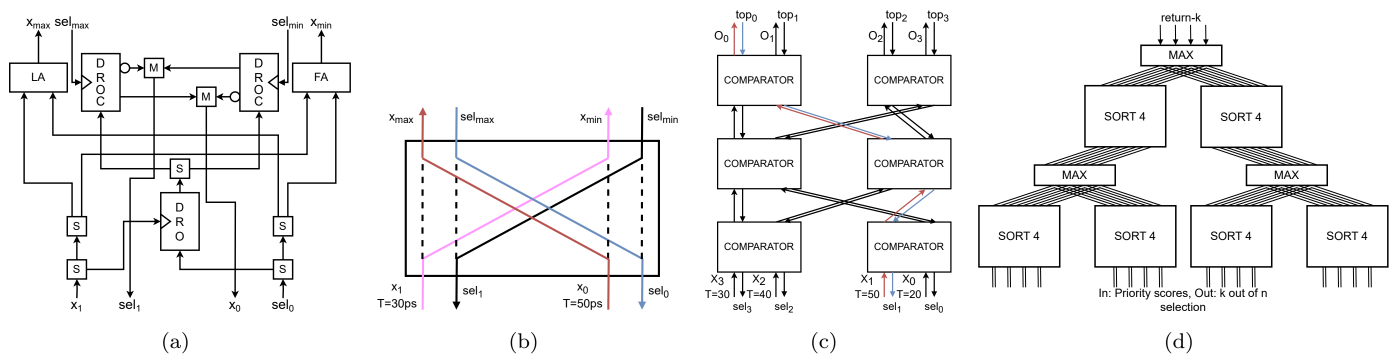

Fig. 1. (a) Circuit diagram of comparator module; (b) Execution of comparator for input times of 30 and 50 ps with signal paths highlighted; (c) Comparators aranged in sorter of size 4 with path of largest priority score highlighted; (d) Arbiter selecting the top 4 out of 16 priority scores

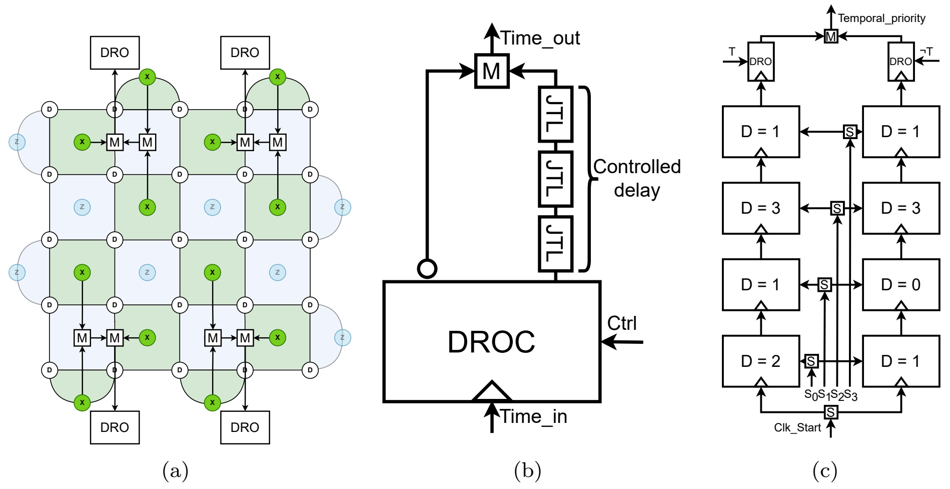

Fig. 2. (a) Merger trees aggregating the complex flags of each quadrant for a surface code of distance 5; (b) Circuit diagram of controllable delay block; (c) Temporal encoder with delay blocks encoding the priority order of table ?

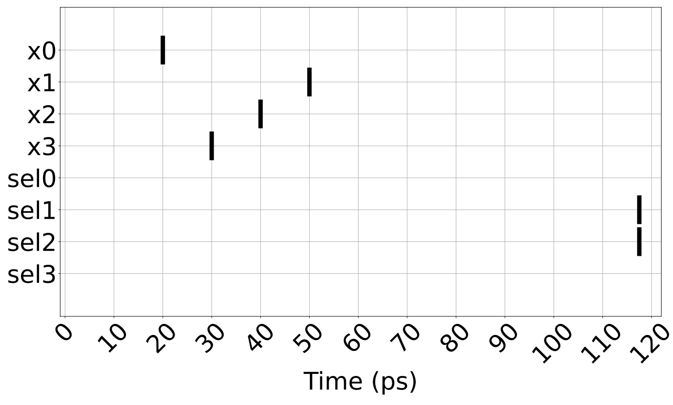

Fig. 3. Simulation of arbiter selecting the two highest scoring qubits out of four. The plot shows the time a pulse (indicated by a vertical bar) appears in each of the arbiter's ports. The score of each qubit is encoded in the arrival time of a pulse in the corresponing input port x0 to x3. After a short delay, the arbiter generates pulses at the sel ports of the qubits with the latest arriving inputs. In this case, sel1 and sel2.