# Combining Perfect Shuffle and Bitonic Networks for Efficient Quantum Sorting

Naveed Mahmud, Bailey Kouson Srimoungchanh, Bennett Haase-Divine,
Nolan Blankenau, Annika Kuhnke, and Esam El-Araby
*Electrical Engineering and Computer Science*
*University of Kansas*, Lawrence, USA
{naveed_923, srimoungchanh.bailey, b.haase-divine, nolanblankenau, akkuhnke, and esam}@ku.edu

***Abstract*—The emergence of quantum computers in the last decade has generated research interest in applications such as quantum sorting. Quantum sorting plays a critical role in creating ordered sets of data that can be better utilized, e.g., quantum ordered search or quantum network switching. In this paper, we propose a quantum sorting algorithm that combines highly parallelizable bitonic merge networks with perfect shuffle permutations (PSP), for sorting data represented in the quantum domain. The combination of bitonic networks with PSP improves the temporal complexity of bitonic merge sorting which is critical for reducing decoherence effects for quantum processing. We present space-efficient quantum circuits that can be used for quantum bit comparison and permutation. We also present a reconfigurable hardware quantum emulator for prototyping the proposed quantum algorithm. The emulator has a fully-pipelined architecture and supports double-precision floating-point computations, resulting in high throughput and accuracy. The proposed hardware architectures are implemented on a high-performance reconfigurable computer (HPRC). In our experiments, we emulated quantum sorting circuits of up to 31 fully-entangled quantum bits on a single FPGA node of the HPRC platform. To the best of our knowledge, our effort is the first to investigate a reconfigurable hardware emulation of quantum sorting using bitonic networks and perfect shuffle.**

***Index Terms*—Quantum Computation, Quantum Emulation, High-Performance Reconfigurable Computing**

## I. Introduction

The field of quantum computing is very promising because of emergence of algorithms such as Shor's factoring algorithm [1], Grover's algorithm [2], Deutsch's algorithm [3], and others [4], [5]. It has been shown that quantum algorithms like these can, in theory, solve NP-hard problems in polynomial time [6]. What used to be thought of as next to impossible in a classical computer may be trivial for a quantum computer to solve. The speedup of quantum systems is largely due to their inherent parallelism [7], contributed by quantum mechanical properties like superposition and entanglement [8]. While the theoretical potential is there, actual physical quantum computers are still under the process of development. Companies like Google [9], IBM [10], D-Wave [11], Riggetti [12], and IonQ [13] are all developing noisy intermediate-scale quantum (NISQ) devices which are capable of processing up to hundreds of qubits. However, it is estimated [14] that a quantum computer needs hundreds of thousands of qubits to be capable of solving real-world problems. The NISQ devices that are being developed right now are also expensive and error-prone [15]. For this reason, researchers have been looking into alternative ways to simulate/emulate quantum hardware on classical computers, particularly using Field Programmable Gate Arrays (FPGAs). FPGA architectures can emulate the parallel nature of most quantum algorithms more accurately than sequential software simulators. In addition, FPGAs are cheaper, faster, and reconfigurable as emulators compared to software simulators. They can be used to verify and benchmark results obtained from actual NISQ systems.

In this work, our objectives are to develop reconfigurable emulation architectures for investigating the use of quantum computers for sorting applications. In the classical domain, there are many algorithms that require sorted data, as it is easier to manipulate than random data [16], [17]. Within the emerging quantum domain there also exist a few quantum algorithms that utilize sorted data, e.g., ordered search and network switching [18], [19]. Sorting is a common and critical operation for large applications [16], and is also suitable for parallel implementations [17]. An efficient algorithm for sorting is the bitonic sort, because of its highly parallelizable structure [20]. This makes bitonic sort suitable for mapping on quantum computers, which are inherently parallel. However, bitonic sort is usually implemented as hierarchical, recursive merge networks [20], [21], and its spatio-temporal complexity of $O(N \log^4 N)$ [20] is not optimal. This makes direct mapping of bitonic merge networks on quantum systems inefficient due to the high temporal cost, which results in high decoherence effects in quantum realizations.

In this paper, we propose combining *perfect shuffle permutation* techniques with *bitonic sort* as a quantum sorting algorithm, to improve the temporal complexity of the sorting and lower decoherence effects for quantum processing. We also present hardware implementation of the proposed quantum sorting algorithm, which is emulated using Field Programmable Gate Arrays (FPGA) on a high-performance reconfigurable computing (HPRC) platform. The hardware architecture is fully pipelined and supports double-precision computations for high throughput and high accuracy. The main contributions of this work are (1) combining perfect shuffle and bitonic networks for quantum sorting, and (2) emulation of the quantum sorting algorithm on an FPGA. To the best of our knowledge, our effort is the first to propose quantum sorting using bitonic networks and perfect shuffle, and to present a

978-1-7281-5999-7/19/$31.00 ©2019 IEEE
DOI 10.1109/H2RC49586.2019.00011

corresponding reconfigurable hardware solution.

The rest of the paper is organized as follows. Section II covers background information on quantum computing. Section III is a survey of related work in existing literature. Section IV contains an overview of our proposed methodology. An in-depth look into our hardware architecture is provided in section V, while experimental results and their analysis are presented in section VI. Finally, section VII concludes the paper and discusses future extensions of this work.

## II. Background

### A. Quantum Bits and Superposition

In quantum computing, the smallest state for storing information is called a qubit. Qubits have two different types of states; computational basis and superimposed [22], [23]. In the computational basis state, a qubit can either be in the 0 state or the 1 state, denoted as $|0\rangle$ and $|1\rangle$ in *bra-ket* notation [22], [23]. In the superimposed state, the qubit state can be written as $|\psi\rangle$ and is equal to a two-dimensional complex vector space as shown in (1).

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \equiv \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{1}$$

where the coefficients $\alpha$ and $\beta$ are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. When a measurement is made on the superimposed state of the qubit, the state collapses to one of the computational basis states, $|0\rangle$ or $|1\rangle$. The probability of finding the qubit in the $|0\rangle$ basis state is $|\alpha|^2$, while the probability of finding the qubit in the $|1\rangle$ basis state is $|\beta|^2$.

### B. Quantum Entanglement

A system with multiple qubits exhibits the unique property of quantum entanglement [22], [23]. A quantum system is entangled if and only if the quantum state cannot be factored into a tensor product of the base qubits. For example, in an entangled system with two qubits, $q_1$ and $q_0$, their joint state, $|\psi\rangle_{q_1 q_0}$, cannot be written as a tensor product of $|\psi\rangle_{q_1}$ and $|\psi\rangle_{q_0}$. In other words, $|\psi\rangle_{q_1 q_0} \neq |\psi\rangle_{q_1} \otimes |\psi\rangle_{q_0}$ [23]. When qubits are entangled, operations on one qubit can effect other qubits. Likewise, measuring an entangled qubit can give information about the state of other entangled qubits. This phenomena, along with qubit superposition, allows for quantum computers to outperform classical computers for certain use cases [22], [23].

### C. Quantum Gates

In quantum computing, quantum gates are the counterpart to logic gates of a classical computer [22], [23]. Quantum gates are used to manipulate qubits and are represented by $2^n \times 2^n$ matrices where $n$ is the number of qubits. For a quantum gate to operate on a qubit, a vector-matrix product is calculated between the gate matrix and the qubit's state vector. Quantum circuits are larger transformations created by the arrangement of a sequence of quantum gates. A few important elementary quantum gates are the X Gate, cX Gate, SWAP Gate, and cSWAP Gate.

*1) X Gate:* A single-qubit gate that swaps the $\alpha$ and $\beta$ coefficients of the input qubit [22], [23].

*2) cX (controlled X) Gate:* A two-qubit gate consisting of a control qubit $|\eta\rangle$ and a target qubit $|\psi\rangle$. If $|\eta\rangle = |1\rangle$, the target qubit will be inverted, otherwise it will remain unchanged [22], [23]. If more than one control qubits are used, the gate is referred to as a $c^kX$ gate where $k$ is the number of control qubits.

*3) SWAP Gate:* A two-qubit gate that simply swaps the position of its input qubits [22], [23].

*4) cSWAP (controlled SWAP) Gate:* A 3-qubit gate that works by swapping the state coefficients corresponding to the second and third qubits when the first qubit, i.e., the control qubit, is set to a state of $|1\rangle$ [22], [23].

### D. Classical and Quantum Sorting

The classical bitonic merge sort is a highly parallelizable, comparison-based sorting algorithm which is generally modeled as a sorting network [24]. This algorithm can sort unordered sequences of *N* elements in $O(N \log^4 N)$ spatio-temporal complexity, by the repeated merging of two bitonic sequences to form a larger sequence [20]. More efficient sorting algorithms have been proposed, such as bitonic sorting with perfect shuffle with spatio-temporal complexity of $O(N \log^2 N)$ [20]. The time complexity, in general, for these classical sorting algorithms is $O(\log^2 N)$ on a parallel machine with $N$ processing elements, and $O(N \log^2 N)$ on a sequential machine [20].

Over the years, limited but quite significant work has been done on quantum sorting [19], [25], [26]. Quantum systems, unlike their classical counterparts, exhibit quantum mechanical properties which contribute to their large degrees of inherent parallelism [8]. Therefore, a quantum bitonic sorter circuit could operate in parallel on an ordered relation defined by $|\psi_0\rangle \leq |\psi_1\rangle \leq |\psi_2\rangle \leq \ \ldots \ \leq |\psi_{n-1}\rangle$ for $n$ qubits. When the input data is encoded as coefficients of a superimposed quantum state [23], mapping the highly parallelizable bitonic sorting algorithm on a quantum computer will improve the sorting spatio-temporal complexity from $O(N \log^2 N)$ on classical systems to $O(n \log^2 n)$ on quantum systems, where $n = \log_2 N$ is the number of qubits. This means quantum bitonic sorting has the potential of achieving exponential speedup relative to bitonic sorting on classical sequential machines.

### E. Perfect Shuffle Based Bitonic Sort

A more efficient implementation of bitonic merge networks on parallel systems is to combine bitonic sort with perfect shuffle [20]. In the perfect shuffle process, two groups of $N/2$ elements are interspersed to form a single group of $N$ elements. For each index *i* such that $0 \leq i \leq (N/2 - 1)$, the element at index *i* of the second group will be preceded by the element at index *i* of the first group [27]. Perfect shuffle features two useful properties. Firstly, for *N* elements, where $N = 2^n$ and *n* being some integer, after *n* iterations of the perfect shuffle process, the sequence of *N* elements

will regain its initial order [20]. Secondly, the index of any new element following a perfect shuffle will be a single cyclical shift to the left of the binary index of the element prior to the shuffle. These properties allow perfect shuffle routing to simplify the hierarchical and recursive structure of bitonic sort [20], resulting in improved temporal complexity. In overview, the sort initially loads a sequence into storage, followed by a perfect shuffle of the sequence elements and then followed by each bitonic comparator receiving two elements of the sequence. Dependent upon the stage of the circuit, the elements will be loaded into the corresponding comparator element. Following the conclusion of all stages, the result will be a sorted sequence in ascending order.

## III. Related Work

As a new technology, quantum computers still suffer from immaturity [28], lack of availability to most researchers [29], and high incurring costs [30], thus creating a focus on quantum simulators/emulators. A variety of parallel software/hardware solutions already exist for the simulation of quantum circuits [31], using technologies such as Graphical Processing Units (GPUs) [32], [33], Central Processing Units (CPUs) [33], and Field-Programmable Gate Arrays (FPGAs) [34]–[40].

Due to the parallel nature of FPGAs they provide an ideal platform for quantum emulation of quantum algorithms which can help reduce emulated resource utilization and time of execution to more accurately emulate the natural complexity of the algorithm. The existing work of quantum emulation on FPGAs have a variety of methods and architectures. In spite of the different emulation methods and architectures, resource overhead and time of execution of quantum algorithms remain the most prominent area of improvement for emulation. For example, in order to improve on the time of execution of emulating a quantum algorithm Pilch and Długopolski [39] proposed a universal architecture that can be used to emulate arbitrary quantum circuits. However, the proposed architecture has a drawback of exponential resource growth with each additional qubit. In addition, the experimental results of their proposed architecture reported a low number of qubits, i.e., 2 qubits, operating at a low, i.e., 10-bit fixed-point precision. In another approach to improving the emulation of quantum algorithms on FPGAs, Frank et al. [40] sought to reduce the resource cost through their proposed Space-Efficient Quantum Computer Simulator (SEQCSim). The reported memory usage indicated a linear growth as the circuit size increased with a small constant decrease in speed of execution. However, their results were obtained from the SEQCSim software prototype rather than from actual hardware implementation on FPGAs of which at the time of publication was being designed. A detailed quantitative comparison of our work with existing work on FPGA-based emulation [34]–[39] is provided in the experimental work section.

In their investigation of quantum switching architectures, Cheng and Wang [19] proposed quantum merge sorting (QMS) as a means of reducing execution time of network switching. They proposed a divide-and-conquer structure that would apply bitonic merge sorting to sort $n$ quantum elements in $O(\log^2 n)$ time at $O(n \log^2 n)$ spatio-temporal complexity. In their proposed QMS design, however, the quantum comparator circuit from which the sort circuit was created was left as a black box with no further details provided. Moreover, no implementation prototypes or experimental results were obtained for their proposed QMS and network switching design. In [25], the authors present a proof showing that the lower bound on spatio-temporal complexity of comparison-based quantum sorting algorithms is $\Omega(n \log n)$. Although their approach, which is based on use of a comparison matrix given as an input oracle, is interesting, they did not provide any quantum circuit realizations, nor any experimental results supporting the proof.

In our proposed methodology, we combine bitonic networks with perfect shuffle permutations to improve the temporal as well as spatial complexity of sorting applications on quantum systems. We design a simple quantum comparator circuit that uses one ancilla qubit to sort two data qubits. Compared to reported related work on quantum sorting, our design is fully deployed on an FPGA and is the first hardware implementation of quantum sorting to the best of our knowledge. In our quantitative comparison of experimental results with related work, we only included FPGA-based emulators as parallel software simulators use relatively large and costly hardware platforms which are not comparable to single-FPGA emulation. Moreover, most of the investigations using parallel software simulators [32], [33] have only demonstrated simulation of simple quantum gates and/or random circuits rather than full algorithms.

## IV. Proposed Methodology

### A. Algorithm

Our proposed algorithm is based on the bitonic sort using perfect shuffle and will sort the data based on the values of the coefficients of the qubits as shown in Algorithm 1 [20] and the corresponding quantum circuit shown in Fig. 1. If the input or output data are in classical representation, a conversion is necessary to encode/decode the classical data to/from quantum data. In this case, a classical-to-quantum (C2Q) and a quantum-to-classical (Q2C) conversion processes are needed [23]. These processes are outside the scope of this work and our focus is on the quantum algorithm rather than on the C2Q and Q2C conversions, which we preserve for future investigations. For data encoded onto $n$ qubits, the algorithm is broken down into $m$ stages where $m = \log_2(n)$. Each stage, $s$, consists of $m-s$ consecutive *quantum perfect shuffle* (QPS) operations followed by $s$ QPS-comparison pairs, resulting in a total of $m$ QPS and $s$ comparison operations, see Fig. 1. Each comparison consists of $n/2$ comparator circuits and applies either a *min-max* or a *max-min* operation over two qubits depending on a set mode. Mode 0 corresponds to a *min-max* operation and mode 1 corresponds to a *max-min* operation. Further explanation of the comparator and QPS operations and their corresponding quantum circuits is provided later in this section.

There are two different patterns that the comparator modes can take. The first, which is applied when $s < m$, alternates between *min-max* and *max-min*, and is represented as [0,1,...]. The second pattern, which is applied only on the last stage ($s = m$), uses only the *min-max* operation for all comparators in this stage and is expressed as [0,0,...]. For example, for an 8-qubit sorter, see Fig. 2, the comparator pattern will start as [0,1,0,1] for all but the last stage. After a comparison is performed, the new pattern will be perfect-shuffled to become [0,0,1,1] for the next comparison. The pattern will then be reset to either [0,1,0,1] or [0,0,0,0] at the beginning of the next stage, depending on the stage number, so that the shuffled permutations do not carry over into subsequent stages.

Algorithm 1: Bitonic sort with perfect shuffle.

```
for s = 1 to m do
    mode(s)
    for i = 1 to m − s do
        QPS(qubits)
    end for
    for i = m − s + 1 to m do
        QPS(qubits)
        comp(qubits, mode)
        QPS(mode)
    end for
end for
```

The algorithm can be broken down into three types of stages using two different comparator patterns. The first stage and middle stages use one pattern, while the last stage uses a different pattern. In the first stage, QPS is applied $m - 1$ times followed by one QPS-comparison pair with the comparators using the [0,1,...] pattern. For a middle stage, $s$ where $1 < s < m$, QPS is applied $m - s$ times followed by QPS-comparison pairs performed $s$ times. The pattern for this stage always starts with the [0,1,...] pattern which is then quantum perfect-shuffled after each QPS-comparison pair is applied to obtain a new pattern to be used by the next QPS-comparison pair of the same stage. The last stage applies QPS-comparison pair $m$ times. However, for this stage the comparison blocks use the [0,0,...] pattern. Perfect shuffling the pattern after each QPS-comparison pair is not needed in this stage as the pattern will remain the same regardless of how many times the perfect shuffle is performed.

### B. Quantum Comparator

As discussed earlier, our quantum comparator has two modes, *min-max* and *max-min*. Therefore, we needed to design a circuit that has two input qubits, $q_1$ and $q_0$, such that the output would either be $q_1$=*min*$(q_1, q_0)$, $q_0$=*max*$(q_1, q_0)$ if the mode was set to 0 (*min-max* mode) or $q_1$=*max*$(q_1, q_0)$, $q_0$=*min*$(q_1, q_0)$ if the mode is 1 (*max-min* mode). A third qubit is needed to act as a control qubit to determine the mode. As it can be seen in Fig. 3, if the mode qubit is set to $|0\rangle$ our comparator is in the *min-max* mode as shown in Fig. 3a, alternatively if the mode is set to $|1\rangle$ our comparator is in the *max-min* mode as shown in Fig. 3b. This is achieved by using a modified c$^2$X gate that uses $q_1$ and $q_0$ as the control qubits and applies an X gate to the mode qubit if $q_1$ is equal

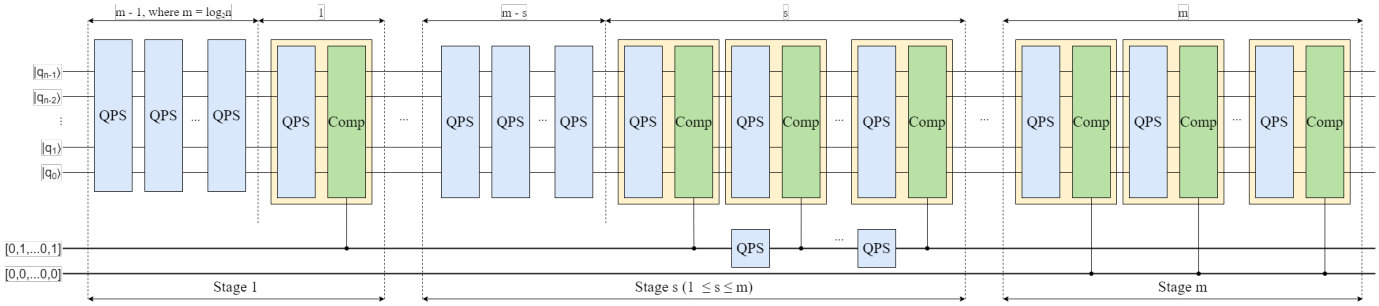



Fig. 1: Quantum circuit of proposed combined bitonic sort and perfect shuffle permutation.

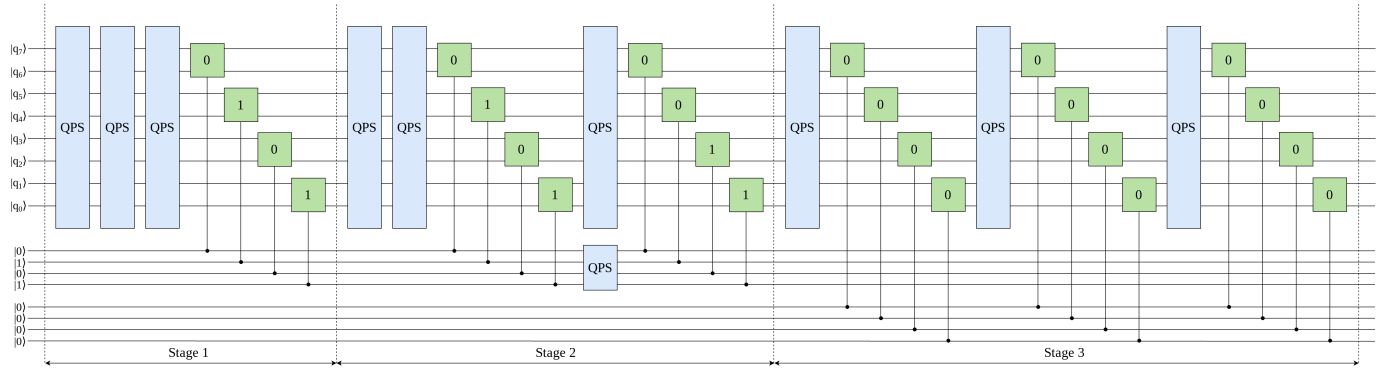
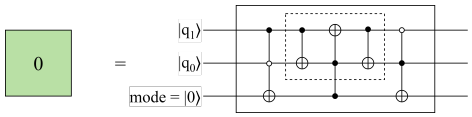
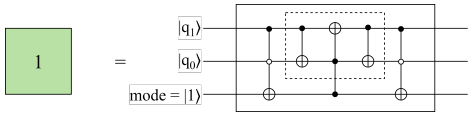



Fig. 2: Example of an 8-qubit quantum bitonic sorter.

to $|1\rangle$ and if $q_0$ is equal to $|0\rangle$. The mode qubit then controls a cSWAP gate that swaps $q_1$ and $q_0$ if enabled. An additional modified $c^2X$ gate is reapplied to reverse the mode qubit back to its original state. For example, if $q_1 = |1\rangle$ and $q_0 = |0\rangle$ and we want to perform a *min-max* comparison we set the mode qubit to $|0\rangle$, the modified $c^2X$ gate inverts the control qubit which becomes equal to $|1\rangle$ enabling the cSWAP gate to swap $q_1$ and $q_0$. Therefore, $q_1$ becomes equal to *min*($q_1$,$q_0$) and $q_0$ becomes equal to *max*($q_1$,$q_0$).




(a) *min-max* comparator for $|0\rangle$ mode.




(b) *max-min* comparator for $|1\rangle$ mode.

Fig. 3: Quantum circuit of algorithm comparator.

### C. Quantum Perfect Shuffle

As discussed in section II, perfect shuffle is used with bitonic networks to efficiently sort data. For our quantum bitonic sort we here provide a quantum circuit that performs the quantum perfect shuffle (QPS) operation of our proposed algorithm. In order to do this, we used a quantum rotate left circuit where all qubits are moved up one position and the most significant qubit is rotated back to become the least significant qubit, i.e., $q_0 = q_{n-1}, q_k = q_{k-1}$ for $0 < k < n$. The quantum rotate left will implement the quantum perfect shuffle since the perfect shuffle rotates indices to the left. The corresponding quantum circuit utilizes cascaded swap gates such that the upper qubits are continuously swapped downward and the lower qubits are swapped upward as shown in Fig. 4.

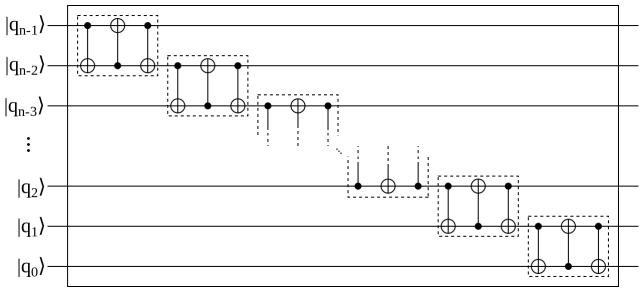



Fig. 4: Quantum perfect shuffle (QPS) circuit.

## V. Hardware Architectures

While designing emulation hardware architectures for the proposed quantum bitonic sorting with perfect shuffle, our target objectives were high space and time efficiency while maintaining high computation accuracy and throughput. For high accuracy, we used double-precision floating-point to model qubits and quantum operations on hardware. The complex parameters of qubits and quantum gate operations were represented using 64 bits each for the real and imaginary components respectively. For example, the data representation used for a single qubit with complex coefficients $\alpha$ and $\beta$ results in a total of 256 bits for each qubit. For high throughput, the design architecture was fully pipelined, which also enabled the circuits to operate at a high frequency, independant of circuit size. For improved space efficiency, the quantum circuit operations were abstracted as classical kernel operations, instead of building hardware models [35] for individual quantum gates of the target quantum circuit. As discussed in Section IV, our proposed algorithm consists of two operations, perfect shuffle and quantum state comparison. We elaborate the corresponding hardware architectures for these operations in the following sections.

### A. Hardware Architecture for Quantum Comparator

The comparator used for quantum state comparison in the proposed sorting algorithm consists of doubly-controlled NOT ($c^2X$) quantum gates and controlled swap (cSWAP) gates. In terms of hardware resources, these gates are costly to implement using a direct quantum gate-based modeling approach [35]. In our proposed architectures, we adopted more space-efficient emulation techniques [41] that significantly improve hardware resource utilization. In our proposed algorithm, it is assumed there exists an accurate classical-to-quantum conversion technique and that the input classical data are encoded as $N$ coefficients of a superimposed quantum state [23]. If the data is encoded only as the real components of the coefficients, the design of the emulator of the quantum state comparator is reduced to the form of a direct classical comparator that compares one pair of state coefficients. In our hardware architecture, we are using double-precision floating-point data, hence a floating-point arithmetic unit is required. The comparison is followed by a simple switching network, which routes the data signals according to the input mode. The emulation architecture for the quantum comparator is shown in Fig. 5.

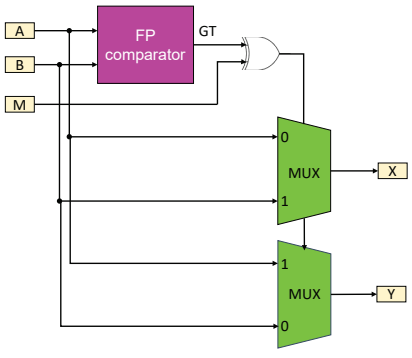



Fig. 5: Architecture for quantum comparator emulator.

The inputs to the quantum comparator emulator are a pair of quantum state coefficients, $A$ and $B$, in double-precision

floating-point format, and a 1-bit control signal, $M$, that selects the mode of the comparison. For *min-max* mode, $M = 0$ and for *max-min* mode, $M = 1$. The internal FP comparator is a typical arithmetic unit [42] that compares two numbers that are in standard IEEE floating-point format, and outputs a *greater than* flag, $GT$. A simple XOR logic gate checks the mode and result of the comparator, and produces the control for routing of the inputs via the MUXes. For emulation, this architecture is simple and space-efficient compared to hardware models of quantum gates such as $c^2X$ and cSWAP.

### B. Hardware Architecture for Quantum Perfect Shuffle

Generally, quantum circuits for perfect shuffle permutations [43], [44] consist of SWAP gates. For classical emulation, the perfect shuffle operation is equivalent to a rotate left operation on the bits of the basis quantum state. In hardware this can be efficiently implemented using index schedulers, that rearrange the sequence of indices of an input vector representing the input quantum state. The shuffled index sequence is then used to write to an output vector representing the output quantum state. A scheduler-based hardware architecture for quantum perfect shuffle is shown in Fig. 6.

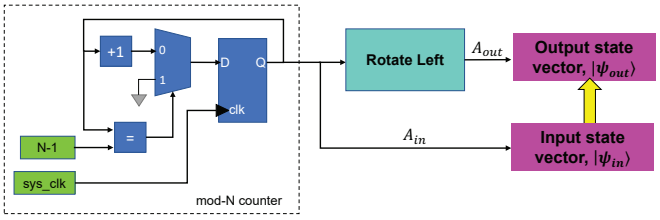



Fig. 6: Emulation architecture for quantum perfect shuffle.

In this architecture, the input and output (shuffled) state vectors occupy separate spaces in memory. A mod-$N$ counter generates a sequence of indices, $A_{in}$, for reading from the input state vector. Each index is rotated left, to produce the output index sequence, $A_{out}$. The output indices are used for writing the read data values into the output state vector, see Fig. 6.

## VI. Experimental Results and Analysis

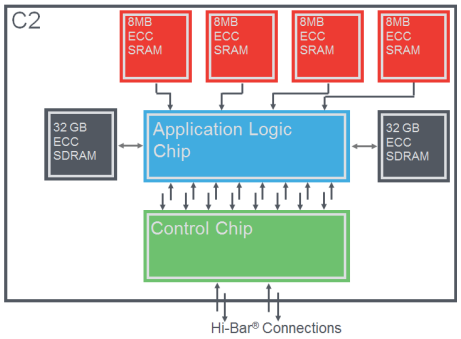



Fig. 7: A compute node on the DirectStream HPRC.

The proposed emulation hardware architectures were implemented on DirectStream (DS8) [45], a state-of-the-art high performance reconfigurable computing (HPRC) system. The DS8 platform supports applications with hardware ranging from single-node compute instances to multi-node instances to multi-chassis racks. The system is independant of any operating system software and is an FPGA-only hardware solution. The DS8 architecture provides benefits such as lower interconnection bottlenecks, less resource contention, and reduced cost and energy use, compared to conventional heterogeneous architectures. A single C2 compute node of the DS8 system is equipped with high-end Intel Arria 10 FPGA and on-board memory (OBM) SRAM and SDRAM modules, as shown in Fig. 7. The DS8 hardware architecture runs on DirectStream's programming environment, which is the successor to the previous compiler, Carte-C [46]. The programming environment facilitates efficient parallel, high-throughput reconfigurable computing through use of a High-Level Language (HLL).

The proposed hardware designs were designed using C++ on DS8. Simulations and hardware implementations were performed using Quartus Prime version 17.0.2 on the Arria 10 10AX115N4F45E3SG FPGA. Reference models developed in Matlab were used to verify correctness of the hardware designs. The designs were fully pipelined, resulting in a system operating frequency of 233 MHz. The on-chip resources on the FPGA consist of 427,200 Adaptive Logic Modules (ALMs), 2,713 Block RAMs (BRAMs), and 1,518 Digital Signal Processing (DSP) blocks, while the on-board memory (OBM) consists of 4×8MB SRAM banks and 2×32GB SDRAM banks.

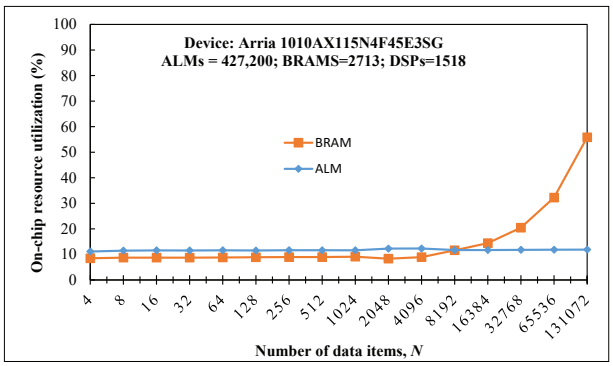
TABLE I: Quantum bitonic sorting using on-chip resources.

| Number of qubits, $n$ | Number of states, $N$ | On-chip* resources | | Emulation time (sec) |
|---|---|---|---|---|
| | | ALMs | BRAMs | |
| 2 | 4 | 47,571 | 230 | 7.74E-06 |
| 3 | 8 | 49,036 | 237 | 2.40E-05 |
| 4 | 16 | 49,460 | 237 | 6.15E-05 |
| 5 | 32 | 49,302 | 237 | 1.54E-04 |
| 6 | 64 | 49,594 | 239 | 3.91E-04 |
| 7 | 128 | 49,253 | 241 | 1.01E-03 |
| 8 | 256 | 49,733 | 243 | 2.85E-03 |
| 9 | 512 | 49,681 | 243 | 8.96E-03 |
| 10 | 1,024 | 49,640 | 247 | 3.09E-02 |
| 11 | 2,048 | 52,400 | 226 | 1.14E-01 |
| 12 | 4,096 | 52,567 | 242 | 4.35E-01 |
| 13 | 8,192 | 50,066 | 315 | 1.70E+00 |
| 14 | 16,384 | 50,078 | 391 | 6.72E+00 |
| 15 | 32,768 | 50,331 | 555 | 2.67E+01 |
| 16 | 65,536 | 50,571 | 875 | 1.07E+02 |
| 17 | 131,072 | 50,768 | 1515 | 4.26E+02 |

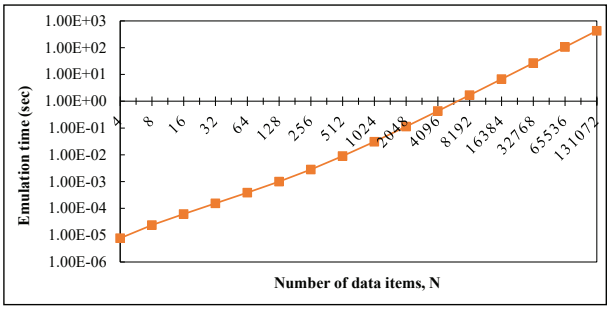
*Total on-chip resources: $N_{ALM} = 427,200$, $N_{BRAM} = 2,713$, $N_{DSP} = 1,518$.

Table I shows implementation results of up to 17-qubit quantum circuits for sorting up to 131,072 items using a single DS C2 node. The limiting factor in the implementation was the Block RAM (BRAM) on-chip resource of the FPGA. The exponential growth of on-chip resources against $N$ is shown in Fig. 8a, where $N$ is the number of data items encoded as the

basis states of the quantum state. The plot of emulation time against $N$ is shown in Fig. 8b. The overall time complexity of emulation was analyzed to be $O(N)$ which includes the I/O time for transferring the quantum input/output state vectors of size $N$ into/from memory. Excluding the I/O time, emulation of the actual quantum sorting operation took $O(\log^2 N)$ time.




(a) Resource utilization.




(b) Emulation time.

Fig. 8: Implementation results using on-chip resources.

TABLE II: Quantum bitonic sorting using on-board memory.

| Number of qubits, $n$ | Number of states, $N$ | On-chip* resources | | OBM** resources (bytes) | |
|---|---|---|---|---|---|
| | | ALMs | BRAMs | SDRAM1 | SDRAM2 |
| 20 | 1.05E+06 | 56,557 | 261 | 8M | 8M |
| 30 | 1.07E+09 | 56,641 | 261 | 8G | 8G |
| 31 | 2.15E+09 | 56,684 | 261 | 16G | 16G |

*Total on-chip resources: $N_{ALM} = 427,200, N_{BRAM} = 2,713, N_{DSP} = 1,518$.
**Total on-board memory (OBM): 4 parallel SRAM banks of 8MB each and 2 parallel SDRAM banks of 32GB each.

To emulate larger circuits, the on-board memory (OBM) resources of the compute node, see Fig. 7, were allocated in the design. In particular, the two 32 GB SDRAM banks were used to store input and output state vectors respectively. Using 64-bit coefficients, this allowed emulation of up to 31 qubits on a single node, as shown in Table II. Our emulation results show that an $n$-qubit quantum sorter is sufficient to sort up to $N = 2^n$ data items with an emulation time complexity of $O(\log^2 N)$ excluding the I/O time. On a quantum computer with $n$ fully entangled qubits, the temporal and spatio-temporal complexities of sorting would be $O(\log^2 n)$ and $O(n \log^2 n)$ respectively. We conclude that bitonic sorting using perfect shuffle is a feasible and efficient methodology that can be adapted for sorting data on future quantum systems.

We provide a quantitative comparison of our work with existing emulators of quantum algorithms in Table III. Among the related work on FPGA emulation of quantum circuits [34]–[39], there has been no demonstration of quantum sorting and we are the first to present a hardware implementation. In addition, our proposed emulator has the capability of emulating the highest number of entangled qubits, i.e., 32 qubits, for a variety of quantum algorithms, operating at the highest frequency, i.e., 233 MHz, and achieving highest accuracy at single/double floating-point precision.

## VII. Conclusion and Future Work

In this work, we investigated the application of bitonic sorting networks with perfect shuffle permutations in the domain of quantum information processing. We proposed a quantum sorting algorithm that employs the aforementioned sorting methodology to sort data items that are encoded as basis states of a superimposed quantum state. We also presented simplified, space-efficient quantum circuits for the proposed sorting algorithm. For prototyping, we have developed an FPGA-based hardware emulator for the quantum sorting algorithm. We designed scalable and high-precision hardware architectures, which were also evaluated experimentally. Our experimental results show that the proposed quantum sorting algorithm is a feasible and efficient methodology that can be adapted in future quantum computers for fast, parallel sorting of data.

Future extensions of this work will be combining quantum bitonic sorting with algorithms such as Grover's quantum search for processing high-resolution, multi-dimensional data in domains such as High Energy Physics (HEP) and remote-sensing hyperspectral imagery. I/O data conversion methods such as C2Q and Q2C will also be investigated in future work.

## References


[1] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Comput.*, 1995.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, 1996. pp. 212-219.

[3] D. Deutsch, "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer," *Proc. R. Soc. A*, vol. 400, no. 1818, pp. 97-117, Jul. 1985.

[4] D. Bernstein. *Post-quantum cryptography*. Springer, Berlin: Heidelberg, 2009.

[5] D. Simon, "On the Power of Quantum Computation," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1474-1483, Oct. 1997.

[6] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 2010.

[7] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings of the thirty-fifth Annual Symposium on Foundations of Computer Science*, 1997. pp. 124-134.

[8] C. Bennett and D. DiVincenzo, "Quantum information and computation," *Nature*, vol. 404, pp. 247-255, 2000.

[9] J. Kelly, "A Preview of Bristlecone, Google's New Quantum Processor," March 2018. [Online]. Available: https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html. [Accessed: August 2019]

[10] IBM Q Experience. [Online]. Available: https://quantumexperience.ng.bluemix.net/qx/devices. [Accessed: August 2019]

TABLE III: Comparison of proposed work with existing work on FPGA emulation

| Reported Work | Algorithm | Number of qubits emulated | Precision | Operating frequency (MHz) | Emulation time (sec) |
|---|---|---|---|---|---|
| Fujishima [34], 2003 | Shor's factoring | - | - | 80 | 10 |
| Khalid et al. [35], 2004 | QFT* | 3 | 16-bit fixed pt. | 82.1 | 61E-9 |
| | Grover's search | 3 | 16-bit fixed pt. | 82.1 | 84E-9 |
| Aminian et al. [36], 2008 | QFT | 3 | 16-bit fixed pt | 131.3 | 46E-9 |
| Lee et al. [37], 2016 | QFT | 5 | 24-bit fixed pt. | 90 | 219E-9 |
| | Grover's search | 7 | 24-bit fixed pt. | 85 | 96.8E-9 |
| Silva et al. [38], 2017 | QFT | 4 | 32-bit floating pt. | - | 4E-6 |
| Pilch et al. [39], 2018 | Deutsch | 2 | - | - | - |
| Proposed work | QFT | 32 | 32-bit floating pt. | 233 | 7.92E10† |
| | QHT** | 30 | 32-bit floating pt. | 233 | 13.8 |
| | Grover's search | 32 | floating pt. | 233 | 7.92E10† |
| | Quantum sorting | 31 | 64-bit floating pt. | 233 | 1.141E11† |

*QFT ≡ Quantum Fourier Transform [37] **QHT ≡ Quantum Haar Transform [41]
†Results projected using regression


[11] D-Wave Systems Inc, "The D-Wave 2000Q$^{TM}$ Quantum Computer Technology Overview," [Online]. Available: https://www.dwavesys.com/sites/default/files/D-Wave 202000Q 20Tech 20Collateral_0117F.pdf. [Accessed: August 2019]

[12] Rigetti Computing. [Online]. Available: https://rigetti.com/. [Accessed: August 2019]

[13] B. Wang, "IonQ Has the Most Powerful Quantum Computers With 79 Trapped Ion Qubits and 160 Stored Qubits," Dec. 2018. [Online]. Available: https://www.nextbigfuture.com/2018/12/ionq-has-the-most-powerful-quantum-computers-with-79-trapped-ion-qubits-and-160-stored-qubits.html. [Accessed: August 2019]

[14] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nat. Phys.*, vol. 14, no. 6, pp. 595-600, June 2018.

[15] L. Gomes, "Quantum Computing: Both Here and Not Here," IEEE Spectrum, April 2018. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8322045. [Accessed: August 2019]

[16] R. Sedgewick, and K. Wayne, *Algorithms*, 4th ed. Upper Saddle River,NJ:Addison-Wesley Professional, 2011.

[17] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing, Second Edition*. Addison-Wesley. 2003.

[18] A. M. Childs, A. J. Landahl, and P. A. Parrilo, "Improved quantum algorithms for the ordered search problem via semidefinite programming," *Physical Review A*, vol. 75, no. 3, Mar. 2007.

[19] S. Cheng and C. Wang, "Quantum switching and quantum merge sorting," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 2, pp. 316-325, Feb. 2006.

[20] S. G. Akl, *Parallel Sorting Algorithms*. Orlando, FL: Academic Press, 1985.

[21] J. Johnson, M. Douze, and H. Jègou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, 2019.

[22] N. S. Yanofsky and M. A. Mannucci, *Quantum Computing for Computer Scientists*. Cambridge: Cambridge University Press, 2008.

[23] C.P. Williams, *Explorations in quantum computing* 2nd ed. Longen: Springer-Verland London, 2011.

[24] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30-May 2, 1968, spring Joint Computer Conference*, pp. 307-314, 1968.

[25] P. Hoyer, J. Neerbek, and Y. Shi, "Quantum complexities of ordered searching, sorting, and element distinctness," *Algorithmica*, vol. 34, no. 4, pp. 429-448, Nov. 2002.

[26] H. Klauck, "Quantum time-space tradeoffs for sorting," In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 69-76, June 2003.

[27] H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Comput.*, vol. C-20, no. 2, pp. 153-161, 1971.

[28] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, vol. 52, no. 4, pp. R2493-R2496, Oct. 1995.

[29] M. Dyakonov, "The Case Against Quantum Computing, IEEE Spectrum: Technology, Engineering, and Science News, 15-Nov-2018. [Online]. Available: https://spectrum.ieee.org/computing/hardware/the-case-against-quantum-computing. [Accessed: August 2019]

[30] J. Hsu, "How Much Power Will Quantum Computing Need?," IEEE Spectrum: Technology, Engineering, and Science News, 05-Oct-2015. [Online]. Available: https://spectrum.ieee.org/tech-talk/computing/hardware/how-much-power-will-quantum-computing-need. [Accessed: August 2019]

[31] J. Miszczak, "List of QC simulators," Quantiki, 10-Feb-2019. [Online]. Available: https://www.quantiki.org/wiki/list-qc-simulators. [Accessed: August 2019]

[32] E. Gutiérrez, S. Romero, M. A. Trenas, and E. L. Zapata, "Quantum computer simulation using the CUDA programming model," *Comput. Phys. Commun.*, vol. 181, no. 2, pp. 283-300, Feb. 2010.

[33] A. Avila, R. H. S. Reiser, A. C. Yamin, and M. L. Pilla, "Parallel simulation of Shors and Grovers algorithms in the distributed geometric machine," in *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2017, pp. 412-419.

[34] M. Fujishima, "FPGA-based high-speed emulator of quantum computing," *IEEE International Conference on Field-Programmable Technology* (FPT 2003), Dec. 2003.

[35] A. U. Khalid, Z. Zilic, and K. Radecka, "FPGA emulation of quantum circuits," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 2004, pp. 310-315.

[36] M. Aminian, M. Saeedi, M. S. Zamani, and M. Sedighi, "FPGA-based circuit model emulation of quantum algorithms," *IEEE Computer Society Annual Symposium on VLSI* (ISVLSI '08), pp. 399-404, IEEE, April 2008.

[37] Y. H. Lee, M. Khalil-Hani, and M. N. Marsono, "An FPGA-Based Quantum Computing Emulation Framework Based on Serial-Parallel Architecture," *Int. J. Reconfigurable Comput.*, pp. 1-18, 2016.

[38] A. Silva, and O. G. Zabaleta, "FPGA quantum computing emulator using high level design tools," *Eight Argentine Symposium and Conference on Embedded Systems* (CASE17), pp. 1-6, IEEE, August 2017.

[39] J. Pilch and J. Długopolski, "An FPGA-based real quantum computer emulator," *J. Comput. Electron.*, vol. 18, no. 1, pp. 329-342, Mar. 2018.

[40] M. P. Frank, L. Oniciuc, U. H. Meyer-Baese, and I. Chiorescu, "A space-efficient quantum computer simulator suitable for high-speed FPGA implementation," *Quantum Information and Computation VII*, 2009.

[41] N. Mahmud, E. El-Araby, and D. Caliga, "Scaling reconfigurable emulation of quantum algorithms at high precision and high throughput," *Quantum Engineering*, vol. 1, no. 2, June 2019. doi:https://doi.org/10.1002/que2.19

[42] Xilinx, "Floating-Point Operator v7.1 LogiCORE IP Product Guide," October, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/floating_point/v7_1/pg060-floating-point.pdf. [Accessed: August 2019]

[43] A. Fijany, and C.P. Williams, "Quantum Wavelet Transforms: Fast Algorithms and Complete Circuits," *arXiv:quant-ph/9809004v1*, Sept. 1998.

[44] L. Hai-Sheng, P. Fan, H. Xia, S. Song, and X. He, "The multi-level and multi-dimensional quantum wavelet packet transforms," *Scientific reports*, vol. 8, Sept. 2018.

[45] DirectStream, LLC. [Online]. Available: https://directstream.com. [Accessed: August 2019]

[46] E. El-Araby, S. G. Merchant, and T. El-Ghazawi, "Assessing Productivity of High-Level Design Methodologies for High-Performance Reconfigurable Computers," in High-Performance Computing using FPGAs, Editors W. Vanderbauwhede, K. Benkrid, Part III, Springer New York, 2013, pps. 719-745.