RTHS: A Low-Cost High-Performance Real-Time Hardware Sorter, Using a Multidimensional Sorting Algorithm

Amin Norollah, Danesh Derafshi, Hakem Beitollahi, and Mahdi Fazeli

***Abstract*— This paper proposes a novel hardware-based multidimensional sorting algorithm and its respective architecture, called real-time hardware sorter (RTHS), for emerging data intensive processing applications where performance and resource conservation are serious concerns. The basic idea behind RTHS is to reduce the hardware complexity of parallel hardware sorting architectures (PHSAs) through a high-performance scalable matrix-based sorting method. The proposed method can also be used for implementing Min/Max queues or finding the largest/smallest records exclusively in the big data application. Implementing the RTHS design on a Virtex-7 field-programmable gate array (FPGA) reveals that the number of lookup tables (LUTs) of the proposed method has decreased by 66.3% and 87.3% compared to the conventional Bitonic sorting network (CBSN) and the state-of-the-art PHSA, respectively. In addition, the number of required registers for the proposed method has decreased by 94.8% compared to the state-of-the-art PHSA.**

***Index Terms*— Bitonic sorting network, field-programmable gate array (FPGA), hardware accelerator, low-cost design, parallel sorting, real-time sorter, sorting algorithm, sorting network.**

I. Introduction

SORTING is one of the fundamental operations in applications such as search [1], [2], database operations [3], [4], radio networks [5], artificial intelligence and robotics [6], scheduling [7], [8], image processing [9], data compression [10], and scientific computing [11]. Sorting has a significant impact on the execution time of the entire system for big data processing [12]. Common sorting algorithms, implemented in software, require a large number of iterations and operating cycles to run on general-purpose processors. By increasing the number of input records, the execution time will be increased as well. A number of sorting techniques have sped up their performance by exploiting the parallelism of multicore processors [2], [13] or GPUs [14]. In recent years, designers have taken interest in designing hardware accelerators using field-programmable gate arrays (FPGAs) [15]–[20]. A sorting hardware unit requires minimal resources to implement, but the number of input records and the width of each record will excessively increase the required resources. Input records are sometimes in the form of binary values, integers, or floating point numbers.

It is important to know the worst case execution time of sorting tasks in hard real-time systems. The success of executing a real-time task depends not only on computational results but also on completion of the task before its deadline [21]. Constant response time is often required for real-time and safety-critical systems [22]. To schedule a set of tasks in an optimal manner, a scheduler has to know the exact execution-time of the tasks.

Schedulers also need to sort their tasks in order of their priority to have access to the most and the least important tasks at all times. This queue structure is called "Min/Max queue," which is often implemented in software. The Min/Max queue is used in several applications such as electronics, environmental, medical, and biological analysis, which requires sorting on data streams produced by sensors [23], [24]. It also proved useful to acquire minimum and maximum values in search [1], [2], data mining [25], and statistical data calculations. In the software implementation of the Min/Max queue, the performance and delay of the sorting operation are variable. Therefore, the intrinsic high performance and constant response time of a hardware implementation are desirable for real-time systems.

Batcher's bitonic sorting circuit [26] is parallel hardware sorting architecture (PHSA) which is based on merge operations. Bitonic sorting is composed of parallel data-independent compare-and-swap (CAS) blocks forming a network. Therefore, bitonic sorting is one of the fastest sorting techniques implemented in hardware. Exploiting parallelism and pipelining in hardware [27] has increased the throughput beyond serial sorting methods in software. The amount of resources required for hardware sorting network depends on the number and type of their CAS blocks.

However, several disadvantages come along with the benefits of bitonic sorting network.

1) This method cannot guarantee that the sequence of records is partially sorted in the expected order in middle stages until the sorting process is completely over. In other words, we cannot determine the process of sorting at the beginning or the middle of the network [28].

Manuscript received October 25, 2018; revised February 4, 2019; accepted March 29, 2019. Date of publication May 8, 2019; date of current version June 26, 2019. *(Corresponding author: Hakem Beitollahi.)*

The authors are with the Department of Computer Engineering, Iran University of Science and Technology, Tehran 16846-13114, Iran (e-mail: a_norollah@alumni.iust.ac.ir; derafshi_danesh@comp.iust.ac.ir; beitollahi@iust.ac.ir; m_fazeli@iust.ac.ir).

Digital Object Identifier 10.1109/TVLSI.2019.2912554

2) Bitonic network consumes a considerable amount of resources to be implemented. As the number of inputs increase, CAS block counts increase as well. Furthermore, the length of the critical path will also be increased and the throughput will be reduced as well.
3) Bitonic sorting has excessive memory requirements, compared to common conventional techniques (e.g., merge sort). Therefore, memory utilization puts a limit on the scalability of the bitonic network [15].
4) The predictability of sorting tasks is important in real-time applications [29]. Changing the number of input records will fluctuate the worst case execution-time of bitonic sorting network.

This paper proposes a novel multidimensional sorting algorithm (MDSA) and its corresponding architecture to greatly reduce the required resources, increase memory efficiency and have a little negative impact on execution time, while the number of input records increases. These features make our solution a worthy replacement to other sorting techniques in real-time systems. The proposed method can also be used for two other applications: 1) to implement Min/Max queues to find the minimum and maximum records quickly and 2) to acquire the largest/smallest records in the big data. In the latter case, we have access to the largest and the smallest records at any time throughout the sorting process.

The rest of this paper is organized as follows. Section II introduces the bitonic sorting network and CAS blocks. Section III investigates different sorting hardware using FPGA. Section IV introduces our novel MDSA. In section V, we discuss the proposed approach and its architecture in detail. Section VI presents the experimental results, and finally, Section VII concludes this paper.

II. Background

A. CAS Blocks

A sorting network is the combination of layers of parallel CAS blocks that leads N disordered M-bit input records to N ordered output records. Each CAS block has two inputs and two outputs. If the inputs are sorted, the records will go straight to the output; otherwise, the CAS block will swap records. CAS blocks are implemented by an $M/2$-bit Comparator and $2 \times M$ 2:1 Multiplexers. Each record is divided into two parts (with an equal number of bits): key field and a data field. The key field is used for comparison and reordering, whereas the data field will be passed through the CAS blocks intact.

We have two types of CAS blocks: increasing and decreasing. Fig. 1(a) shows an increasing CAS block. After comparing two entry records, the smaller record and the larger record will pass through the right and left multiplexers, respectively. Using increasing CAS blocks makes for an increasing sorting network with the smallest records on top. Fig. 1(b) shows a decreasing CAS block. This block passes the smaller record through the left multiplexer and the larger record through the right multiplexer. This block makes for a decreasing sorting network with the largest records on top. Fig. 1(c) shows a dual-mode CAS block. As the name implies, this block is designed to work in both ascending and descending order. If the "Direction" signal value is zero, the dual-mode block behaves like an increasing CAS block; otherwise, it turns into a decreasing CAS block.

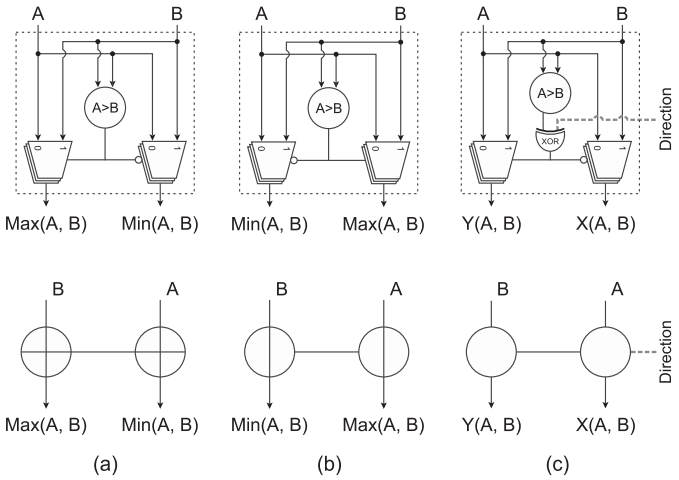

Fig. 1. High-level implementation (top) and schematic symbol (bottom) of building CAS blocks for sorting networks. (a) Increasing CAS block. (b) Decreasing CAS block. (c) Dual-mode CAS block.

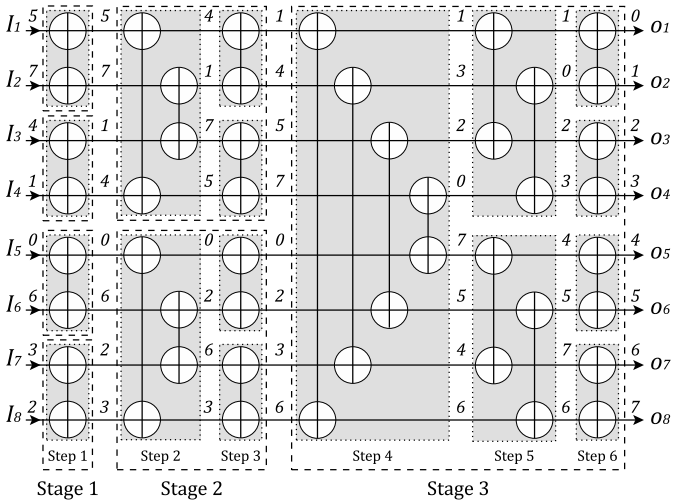

Fig. 2. CAS network for an 8-input bitonic sorting unit with 3 CAS stages, 6 CAS steps, and 24 increasing CAS blocks.

The delay of a CAS block depends on the comparator and the 2:1 multiplexer that are serially connected together. We should add the delay of an XOR gate to this value for a dual-mode CAS block.

B. Bitonic Sorting Network

As mentioned earlier, a series of parallel CAS blocks will form a sorting network. Each time, a set of N unordered records enters the network. CAS blocks sort their input records at each step and send their outputs as inputs to the next step. Bitonic sorting network has been widely used for hardware implementations [9], [15], [16]. In an N input bitonic network, we have $\log_2(N)$ CAS stages. Each K-record stage has $\log_2(K)$ CAS steps with $N/2$ parallel CAS blocks in each step. Fig. 2 shows an eight-input bitonic sorting network. In the first stage, we have four parallel CAS blocks with two records as inputs, and thus, we only need one step to sort them out. Stage 2 consists of two consecutive steps. Each step

receives records from its previous step and sorts the records according to the predefined connections of its corresponding CAS units. Stage 3 includes three steps and its last step outputs the final sorted sequence of the given records. An N-input bitonic network has a total of $1/2\log_2(N)(\log_2(N)+1)$ steps. The total number of CAS blocks required for the network is calculated from the following equation:

$$\text{Num}_{\text{CAS}} = N/4 \times \log_2(N)(\log_2(N)+1). \quad (1)$$

For example, the eight-input bitonic sorting unit shown in Fig. 2 consists of $\log_2(8) = 3$ stages, $1/2 \times \log_2(8)(\log_2(8)+1) = 6$ steps, and $8/4 \times \log_2(8)(\log_2(8)+1) = 24$ increasing CAS blocks. The resources required to implement a bitonic sorting unit depend on the number of CAS blocks. Also, the delay of the network depends on the number of steps. Moreover, according to (1), this delay limits the feasible number of input records.

III. Related Work

Several PHSA solutions have been proposed to enhance the sorting process based on bitonic and merge sorting networks. In this section, we introduce some of these methods that are implemented on FPGAs.

Srivastava *et al.* [15] have proposed a hybrid design for large-scale sorting on a FPGA. Merge sorting network benefits from a small delay and optimal memory utilization, but it suffers from low throughput due to the lack of parallelism in the last stage of its network. On the contrary, the bitonic sorting network achieves a great throughput, at the cost of high delay and memory requirements. The authors managed to blend merge and bitonic sorting networks to avoid their inherent shortcomings. Thus, the initial steps of the sorting network utilize merge sorting structure, while the final steps make use of bitonic method to increase parallelism. This combination improves the throughput and reduces memory consumption, at the cost of high resource requirements. It also needs to sort the entire records to find the minimum and maximum entries.

Ricco *et al.* [30] have introduced a new application of sorting networks on modular multilevel converters (MMC), by presenting a new bitonic-based sorting architecture to enhance the control of capacitor voltages balance (CVB). The authors have presented a factorization method to share the CAS units between factorized submodules. This method decreases resource consumption at the cost of increasing the factorization level. However, the increase in factorization level worsens the delay and the throughput of the sorting network.

The Rocket queue architecture [22] has implemented the Min/Max queue structure based on shift registers and the heap sort. The shift registers are divided into sections with only one CAS block for each section, which reduces the area and power consumption. Incoming records will be compared to the first cell of each queue. If it is smaller, it will be swapped in, else it will move on to the lower sections, and this process continues until it finds a fit for the input record. The critical path of finding the smallest record is fixed and highly predictable. On the other hand, if a new record enters a queue and it happens to be bigger than the majority of enqueued records, it will impose a great delay on sorting process to fit the record which is the main disadvantage of this architecture.

Song *et al.* [16] introduce parallel merge tree (PMT), a high-performance merge sorting network with multirate merger (MM) units at each stage. An MM unit consists of a P input partial bitonic network and P FIFO queues. The partial bitonic network selects $P/2$ of the biggest records and sends them to its outputs. Their architecture achieves high throughput for a small number of records. However, increasing inputs will extend the critical path and will diminish the resource and memory efficiency.

The complexity and throughput of a sorter have a direct impact on power consumption. Lin *et al.* [17] offer a high-performance, low-power sorter architecture. Their solution benefits from a low-power sorting module followed by an adaptive clipping operation, in order to increase throughput. The critical path lengthens slowly as the input records increase, which satisfies real-time constraints, but the resources required for the sorter will increase dramatically.

SHMS [18] is a hardware merge sorter that arranges the sorted segments of records into a general sequence. As mentioned earlier, merge sort takes a toll on frequency due to the increase in the critical path. SHMS tries to solve this problem in an innovative way by circumventing the number of consecutive circuit gates to a constant number, resulting in a constant frequency. This architecture has been able to improve throughput compared to other PHSA solutions.

The low speed and high resource requirements of bitonic and even–odd sorting networks, in the absence of pipelining, have persuaded Sklyarov and Skliarova [31] to devise a sorting method that sorts a set of records in an iterative manner. Iterative sorting includes subsorters, consisting of two layers of consecutive CAS units. The subsorter is fed through registers and eventually its outputs are returned to the same registers. We call this method reusable sorting network (RSN). The authors emphasize that by employing software presorting and partial sorting of incoming records, the number of iterations can be reduced. This method sorts partial sets of records independently and then merges all sets into a single sequence of sorted records, using a merge sort method, implemented in software. Of course, with increasing the number of inputs, the number of iterations will increase as well and the throughput starts to decline.

In [9], unary processing is used to design a low-cost and fault tolerant sorter. Unary processing is a subcategory of stochastic computing. All bits possess equivalent values. Numbers are represented as a bitstream, which allows the computational circuits to be simple and efficient in terms of power and area consumption. CAS blocks are implemented, using only one AND and one OR logic gates. Therefore, the delay and resources required for each CAS block are dramatically decreased. However, their method suffers from low throughput and extensive delays. The latter reacts poorly to a slight increase in data width.

IV. Multidimensional Sorting Algorithm

As discussed in the previous sections, factors such as critical path, resources, and power consumption limit the feasible

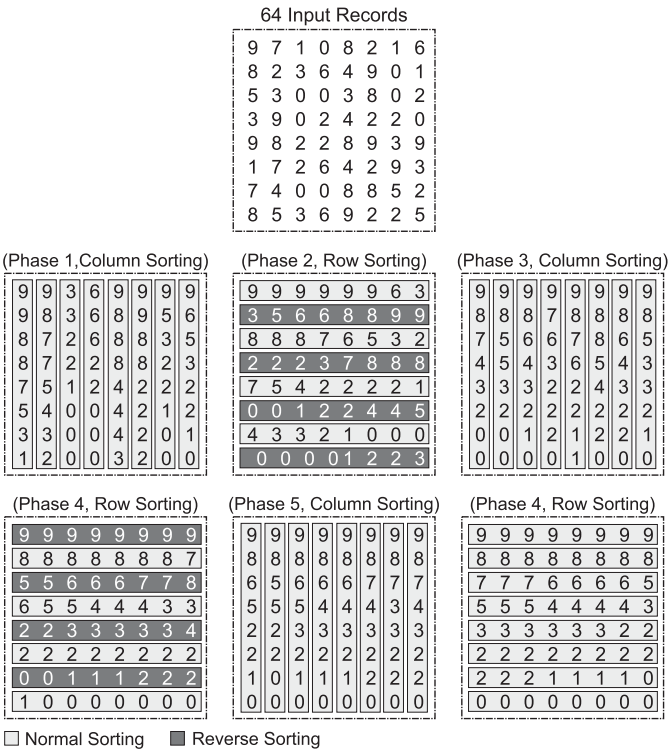

Fig. 3. Two-dimensional proposed algorithm for sorting 8×8 matrix input records.

number of input records. To sort a large sequence of records, the sequence has to be divided into smaller segments and each segment has to be sorted, separately. In the end, the sequence of the sorted records is obtained by a secondary sorter, which is implemented, using a set of FIFO queues and a comparator unit [18]. A secondary sorting circuit imposes a great delay on the system and occupies a considerable amount of resources on an FPGA. The goal is to provide a method that can handle more input records without using a secondary sorter.

Assume N input records with M-bit width form a $P \times P$ matrix ($P = (N)^{1/2}$). The MDSA uses P input sorting networks to sort all records. Fig. 3 shows MDSA in action with 64 input records, forming an 8×8 matrix.

The sorting network can operate in two modes: 1) Normal sorting is the case in which the sorting network arranges records in descending order and 2) reverse sorting is the case in which the sorting network arranges records in ascending order. MDSA has six consecutive phases, as follows.

1) *Column Sorting:* Odd and even networks have normal sorting modes.
2) *Row Sorting:* Even networks have reverse sorting mode, and odd networks have normal sorting mode.
3) *Column Sorting:* Odd and even networks have normal sorting modes.
4) *Row Sorting:* Odd networks have reverse sorting mode, and even networks have normal sorting mode.
5) *Column Sorting:* Odd and even networks have normal sorting modes.
6) *Row Sorting:* Odd and even networks have normal sorting modes.

At each phase, each sorting network sorts its assigned eight records, independently. In phase 1, column records are assigned to the sorting networks. Sorting networks operate in the normal sorting mode and arrange their records in the descending order for each column. In phase 2, row records are assigned to the sorting networks. Odd and even networks are in the normal sorting and reverse sorting modes, respectively. Moving records in adjacent rows, using two normal and reverse sorting modes, makes it possible to compare all the records together in the following phases. In phase 3, sorting networks receive and sort records as columns in the normal sorting mode. After phase 3, partial sorting is done and the larger and smaller records are placed in the upper matrix and lower matrix elements, respectively. Phase 4 reiterates phase 2, and phase 5 is similar to phases 1 and 3. In phase 6, records are assigned to the sorting networks as rows and they will be arranged in normal sorting mode, one last time. At the end of phase 6, all records are sorted and will be sent to the output as a 1-D array.

We tend to use MDSA to sort an n-dimensional matrix. Let $a_{(i,j,\ldots,n)}$ be a matrix element and $i \times j \times \ldots \times n$ represent their corresponding sizes.

Theorem 1: Records in an n-dimensional matrix can be arranged into a 1-D sorted array, if the following relations are established ($1 \leq w \leq n$).

1) In wth dimension, record a_m, must be greater than record $a_{(m+1)}$, for $1 < m \leq P$ and $P = ([n]N)^{1/2}$.
2) In wth dimension, the record $a_{(c,P)}$ of this dimension must be greater than record $a_{(c+1,1)}$, for $1 < c \leq P$. in $a_{(k,l)}$, k is $w-1$th dimension and l is wth dimension.

Proof: If we want to place the records of matrix $A_{(i,j,\ldots,w)}$ in 1-D array B, we will use the following equation:

$$B[P^{n-1}w + \ldots + Pj + i] = A_{(i,j,\ldots,w)}. \tag{2}$$

In (2), $i, j, \ldots, w$ can change in the range between 1 and P. The following equation is used to check the descending order of records in array B:

$$B[m] \geq B[m+1], 0 < m < P. \tag{3}$$

According to (2), in each dimension, records have to be arranged in descending order and the last record of each dimension is larger than the first record of the next dimension. □

Consider set $\{\bar{x}, \underline{x}, x\}$ as the representation of sorting modes in dimension x. In mode x, all sorting networks operate in the normal sorting mode. In mode $\bar{x}$, the odd/even sorting networks operate in the normal/reverse sorting modes, and in $\underline{x}$, the odd/even sorting networks operate in the reverse/normal sorting modes. For example, to sort a 2-D matrix (i represents the row and j represents the column), we require six phases, which are represented as $\{j\bar{i}, j\underline{i}, ji\}$.

We assume a 2-D matrix with the size of $P \times P$, $a_{(i,j)} < a_{(i+1,j)}$ and $a_{(i,j)} < a_{(i,j+1)}$. In phase 1 ($\{j\}$), the matrix columns are arranged in descending order and the $P/2$ larger/smaller records of each column are placed on the top/bottom of the matrix. In phase 2 ($\{\bar{i}\}$), the matrix odd/even rows are arranged in the descending/ascending order.

The record $a_{(P,P)}$ is transferred to the first row and first column of the matrix after this phase, but the record $a_{(P,P-1)}$ remains in row 2 and column 4. These replacements cause the columns with the largest records (records residing in columns P/2 to P), to broadcast their records across the matrix. In other words, it will enable us to compare all the records in the next phases. In phase 3 ($\{j\}$), columns will be arranged in the decreasing order for the second time, and as a result, the upper/lower records of the matrix will be the largest/smallest records. At this point, all records are arranged in their rows and columns, but we tend to send the sorted records as a 1-D sorted array to the output. To respect the conditions of Theorem (1), the following equation is obtained for 2-D sorting ($0 < i \leq P$):

$$\begin{cases} a_{i,j} \geq a_{i,j+1}, & 0 < j < P \\ a_{i,P} \geq a_{i+1,1}, & j = P. \end{cases} \tag{4}$$

The next phases are intended to meet this condition. In phase 4 ($\{\underline{i}\}$), the matrix odd/even rows are arranged in the ascending/descending order. The next two phases ($\{ji\}$) will arrange the rows and columns of the matrix in the descending order. At the end, the condition of (4) is met, and the sorting process is completed.

We can sum up the sorting phases for an n-dimensional matrix with the following equation, which has distributive property, but lacks Commutative property ($q = n - 1$):

$$\{n\{\bar{q}, \underline{q}, q\}\} \ldots \{\bar{k}, \underline{k}, k\}\}\{\bar{j}, \underline{j}, j\}\}\{\bar{i}, \underline{i}, i\}\}. \tag{5}$$

In (5), the sorting modes are always changing in each dimension except for the last one, in which the sorting mode has to be always fixed. To sort an n-dimensional matrix, we need $n \times 3^{n-1}$ sorting phases. For instance, it requires 6 and 27 phases for 2-D and 3-D matrices, respectively. Increasing the phases will extend the delay. However, the number of CAS blocks required to implement a high number of phases will be considerably lower than the conventional solution, presented by (1). If the number of input records remains constant in the main sorter, the sorting network units will be simplified by increasing the dimensions of the matrix.

Fig. 4 shows the sorting phases of MDSA for a 3-D matrix $A_{4,4,4}$, intended for sorting 64 input records. According to (5), it takes 27 phases to sort a 3-D matrix. In the first dimension ($\{i\}$), the sorting mode will be changed for each row, as shown in Fig. 4. In the second dimension ($\{j\}$), the sorting mode will be changed per $\{ji\}$ page and the columns of each page have the same sorting mode. In the third dimension ($\{k\}$), the sorting mode is normal and constant for all phases. The 3-D matrix sorting process can be replaced by the 2-D sorting process shown in Fig. 3. Choosing 3-D over 2-D reduces hardware resources but imposes 4.5 times more sorting phases to the process.

This sorting algorithm eliminates the need for secondary sorting hardware. Moreover, increasing the number of input records does not affect the number of phases. It will affect the size of the input matrix and will increase the number of sorting networks and their number of corresponding stages, but it will not change the phases of the algorithm (six in the 2-D matrix). If we integrate the pipeline stages into sorting networks, we obtain a fixed execution time, so that a real-time scheduler can easily calculate and schedule execution time of sorting tasks. Note that this algorithm is also applicable for other nonsquare matrices. However, the square matrix is the optimum form for hardware implementation. If the increased number of input records is in the range between $2^{2k} + 1$ and $2^{2(k+1)}$ for $k = 1, 2, 3, \ldots$, we will use a sorter, capable of sorting $2^{2(k+1)}$ records. As a result, we will not experience any increase in delay, if the number of records increases in this range. For example, the delays in sorting the number of records between 17 and 64 are the same. However, if the increased number of records exceed $2^{2(k+1)}$ limit, we experience a gradual increase in overall delay, due to the increase in the number of stages of sorting networks.

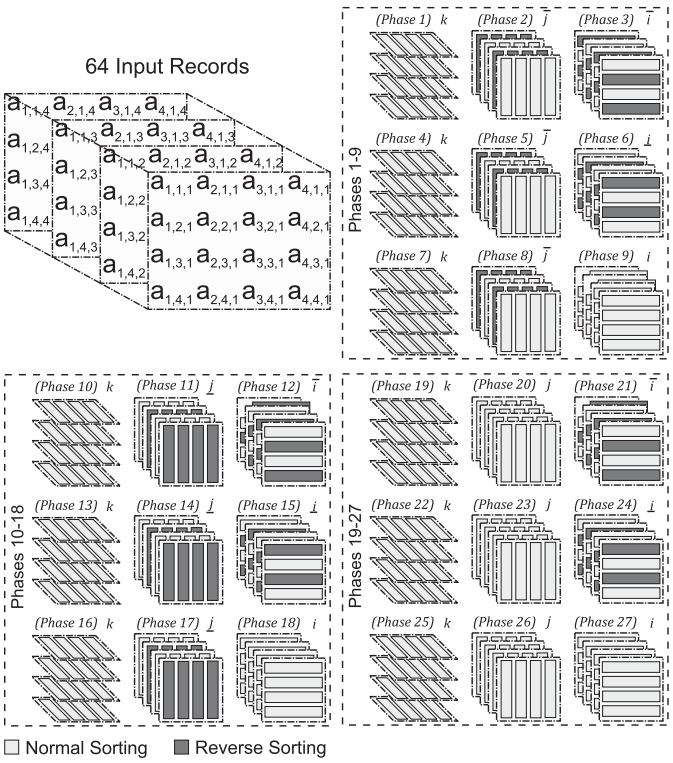

Fig. 4. Three-dimensional proposed algorithm for sorting $4 \times 4 \times 4$ matrix input records.

In big data sorting, we assume a software tool to split the input records into segments and then each segment to be sorted by MDSA. The software tool will assign records to segments based on sorting algorithms such as bitonic. The sorting process continues until all records are sorted.

The second application of this algorithm is to obtain Min/Max records in two phases. According to Fig. 3, at the first phase, the biggest records are in the first row and the smallest records are in the last row. At the second phase, the biggest record is in the first row and first column and the smallest record is in the last row and first column.

V. Proposed Approach

In this section, we describe the hardware implementation of MDSA for a 2-D input matrix. To illustrate our design,

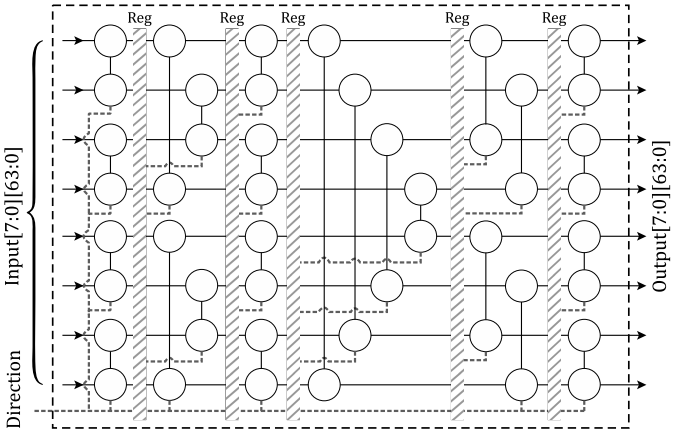

Fig. 5. Eight-input DPBN.

we first discuss the design of dual-mode pipeline bitonic network (DPBN).

A. Dual-Mode Pipeline Bitonic Network

Increasing the number of input records in a bitonic sorting network will increase the number of steps and will further deepen the network. This will lengthen the critical path and overall network latency. Therefore, pipelining is used as a technique to reduce the critical path.

The number of pipeline stages in a DPBN is equivalent to the number of its steps $(1/2\log_2(N)(\log_2(N)+1))$. Fig. 5 shows an eight-input DPBN. The network can receive up to eight input records with 64-bit width per clock and deliver sorted records to its output after six stages of the pipeline. The control unit sends the "direction" signal to the DPBN to set mode between the normal and reverse sorting.

B. Main Proposed Sorting Hardware

According to the proposed algorithm for a 2-D input matrix, we need P column sorters and P row sorters. We also need the row sorters to include both decreasing and increasing CAS block types. We design the total sorting network with P units of DPBN to reduce the hardware requirements. Fig. 6 shows the proposed sorting architecture called real-time hardware sorter (RTHS) for sorting 8×8 matrix records. It is composed of eight DPBNs, an 8×8 implicit switch, registers for storing records, and a controller. As shown in Fig. 7, the implicit switch is implemented, using fixed wiring. The multiplexing circuit is implemented by an implicit switch. The switch will interchange the position of rows and columns in a matrix. The switch is implicitly implemented within the sorting network and does not require additional hardware (only requires more complex routing in the FPGA). In Fig. 7, each horizontal line is connected to an input line, carrying a set of eight 64-bit records. Each horizontal line then distributes each of its records to eight vertical output lines. All the first records of all inputs will be merged into a set of eight records and will be sent out through the first output. This transition is similar across all the outputs. Note that the process of breaking down and concatenating the sets of eight records is completely static and only imposes a minimal overhead to the routing of the FPGA, due to its intrinsic hardwiring.

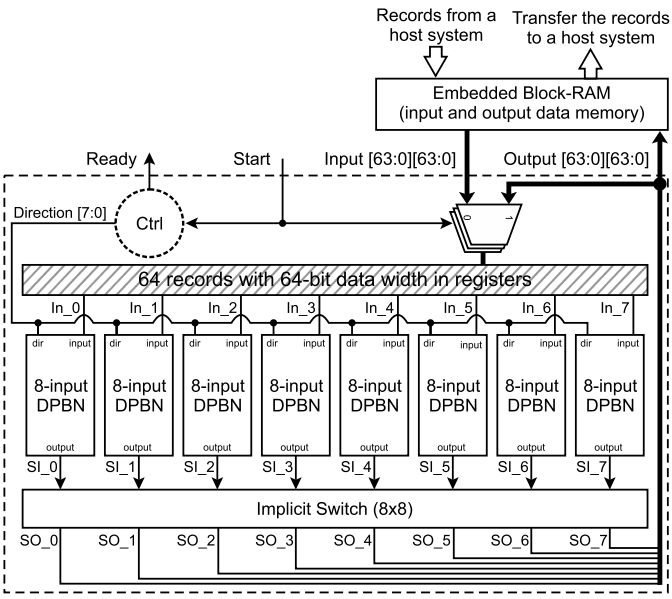

Fig. 6. RTHS design for sorting 8×8 matrix records.

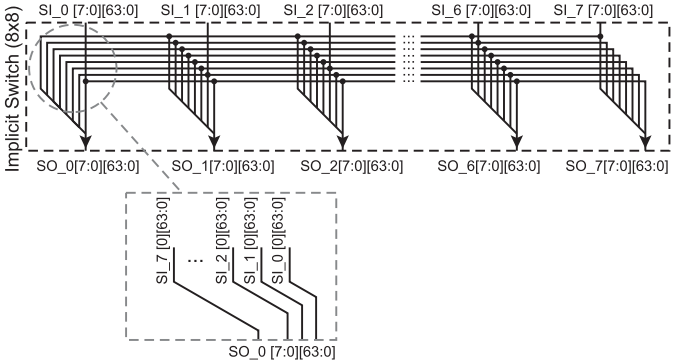

Fig. 7. Implicit switch design.

At the first, the process is initialized by loading 64 records with 64-bit data width into the primary registers from the block random access memories (BRAMs) and next "Start" signal is activated. In the next cycle, the records of the primary registers are assigned to eight DPBN units for sorting. The control unit sends out the "Direction" signals to the DPBN units according to the proposed sorting algorithm. Each bit of the "Direction" signal determines the sorting mode of its corresponding DPBN unit. "0" value represents normal sorting, while "1" value indicates the reverse sorting mode. The switch swaps rows and columns of the matrix and sends them back to the primary registers. The control unit manages the execution of the six-phase sorting process by issuing proper control signals. In the end, the control unit will issue the "Ready" signal to declare that the records residing in the primary registers are sorted.

Fig. 8 shows the finite-state machine (FSM) of the control unit. The sorter is initially in the "Wait" state. An initial value will be assigned to the "Direction" signals when the "Start" signal is activated. After every six clock cycles (number of pipeline stages in DPBNs), the "Delay" signal (counts delays) will be activated. The FSM states and the "Direction" signals will be changed according to Fig. 8. Finally, as the "Ready"

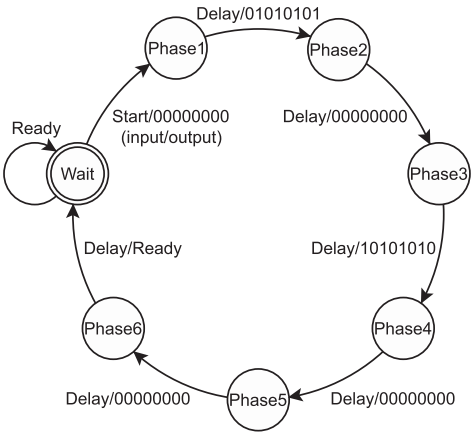

Fig. 8. FSM of the control unit.

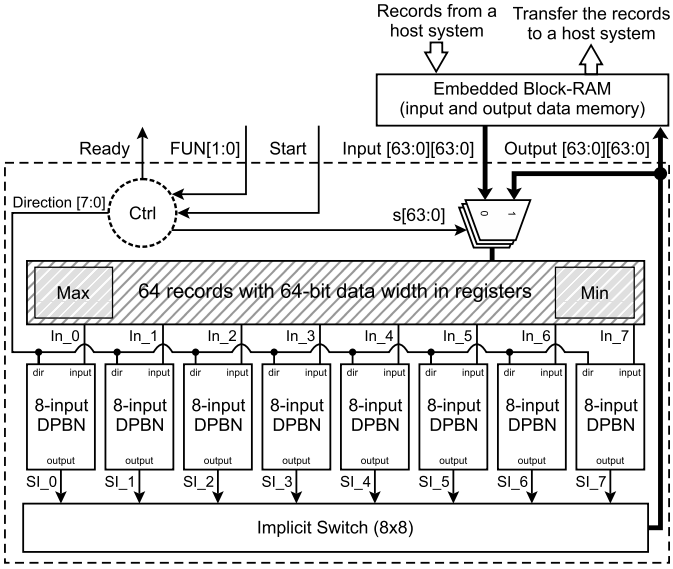

Fig. 9. Improved RTHS design.

signal becomes activated, the sorter will return to the "Wait" state.

C. Extended Proposed Sorting Hardware

Some applications need to know the maximum and minimum records as fast as possible. As shown in Fig. 3, it only takes two phases of the proposed algorithm to find the maximum and minimum records in a 2-D input matrix. Fig. 9 shows an improved architecture for obtaining the minimum and maximum records. Initially, 64 input records with 64-bit width will be allocated to the sorter. After two phases, the maximum and minimum records are in the leftmost and the rightmost registers, respectively. Then, the "Ready" signal will be activated and the midlevel records will be replaced by 62 new records in primary registers. This process will continue until all big data records are fed to the sorter. In the end, the two most lateral registers are holding the minimum and maximum records of big data.

In general, the proposed method can obtain $P/2$ maximum records and $P/2$ minimum records after two phases, because, at the end of phase 2, the first and last rows contain the largest and smallest records, respectively. Note that, at this point, only half of the P records, residing in the first and last rows, are in the correct order. The overall delay will be increased due to the reduction of entry inputs to $(N - P)$ record at each replacement.

The architecture shown in Fig. 9 can also be used for Min/Max queues. Assume that we have relative deadline values of each real-time task in the key field of the input records. When a new task arrives at this queue, the task with a maximum relative deadline will be swapped out for the new task to replace it. However, if the relative deadline of the new task is larger than the maximum one on the Min/Max queue, it will not be added to the queue after all. The Min/Max queue will be updated in two scenarios: 1) a new task is accepted and 2) the task with a minimum relative deadline is assigned to a processor to run.

The sorter can be used for three different applications only by changing the controller modes: 1) general sorting; 2) finding the minimum and maximum records in big data; and 3) creating Min/Max queues. Fig. 9 shows the overall RTHS architecture, with the ability to change the applications of the sorter by receiving different "FUN" signals. The general sorting mode is indicated by "00," which means that all input records will be sorted after six phases. If FUN = "01," the sorter will find the minimum and maximum records in big data and finally, if FUN = "10," the sorter will switch to the Min/Max queue mode, in which only the largest record in the queue will be swapped out under certain circumstances. The maximum and minimum records will be sent to the output at once.

D. Proposed Sorting Analysis

The overall delay of the sorting network relies on the maximum delay of a CAS-Dual unit and the implicit switch routing in the FPGA. Therefore, if the implicit switch routing is ideal, the delay of the sorter is almost constant. The memory usage is also important and relies on three factors: 1) the number of DPBN units; 2) the number of pipeline stages; and 3) the registers of each pipeline stage.

The number of registers required per pipeline stage is obtained from the following equation:

$$\text{block}_{\text{reg}} = M \times P \tag{6}$$

where M is the data width and P is the number of records in a row or column. The number of registers required by a DPBN unit is obtained from the following equation:

$$\text{bitonic}_{\text{reg}} = \text{block}_{\text{reg}} \times (1/2 \times \log_2(P)(\log_2(P) + 1)). \tag{7}$$

Finally, the total registers of the RTHS network will be calculated from the following equation:

$$\text{total}_{\text{reg}} = (\text{bitonic}_{\text{reg}} \times P) + M \times N. \tag{8}$$

The number of required registers for the conventional bitonic sorting network (CBSN) is obtained from the following equation:

$$\text{CBSN}_{\text{reg}} = M \times N \times (1/2 \times \log_2(N)(\log_2(N) + 1)). \tag{9}$$

TABLE I
COMPARISON OF DESIGN FOR SORTING N ELEMENTS

Design	Proposed	Conven. Bitonic
Latency	$Max(delay_{CASDual}, Routing)$	$\log_2 N \times delay_{CAS}$
Memory	$M \times P^2 \times \log_2^2(P)$	$M \times N \times \log_2^2(N)$
Throughput	N	N

Therefore, the total number of registers required for the RTHS network is of order of $O(M \times P^2 \times \log_2^2(P))$, while this number is of order $O(M \times N \times \log_2^2(N))$, for the CBSN. To sort 64 records with 64-bit width, the proposed method requires [(from (8)] ([$(64 \times 8) \times (1/2 \times \log_2(8)(\log_2(8) + 1))] \times 8) + 64 \times 64 = 28\,672$ registers, while CBSN needs $(64 \times 64) \times (1/2 \times \log_2(64)(\log_2(64) + 1)) = 86\,016$ registers [from (9)]. Therefore, we can conclude that RTHS uses less memory compared to the CBSN solution. We also assume the number of pipeline stages of CBSN, to be equal to the number of stages in its network, for a fair comparison between CBSN and RTHS.

Table I compares our design against CBSN. The CBSN latency relies on the depth of the sorting network, which translates to the number of consecutive CAS blocks between two stages. Routing in Table I is referred to as the delay of routing on the FPGA.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

A. Experimental Setup

We have implemented the RTHS design on a Virtex-7 FPGA (XC7VX485T, speed grade −2L), written in Verilog hardware description language. This device has 360 dual-port BRAMs (each 36 Kb) and 303 000 CLBs. We also used Vivado-2018.2 [32] for synthetization and place-and-route purposes. In the test cases which demand resources beyond the FPGA limits, we reported results, obtained in the synthesis phase. The metrics are as follows.

1) *Resource Utilization:* The number of resources required for the sorter, such as lookup tables (LUTs), registers and operating frequency.
2) *Throughput:* Number of sorted bytes per time unit (gigabytes per second).
3) *Execution Time:* Time required to sort one sequence of input records.
4) *Memory Efficiency:* The achieved throughput divided by the on-chip memory consumption (in bits). We are specifically interested in the top-left area of the memory efficiency plot that indicates the most efficiency coupled with the least memory usage.
5) *Power Consumption:* The amount of power in watts per time unit.

B. Resource Utilization Analysis

In this section, the resource utilization of the RTHS design is compared to CBSN and other common methods. Table II and Fig. 10 illustrate the results of comparing RTHS against CBSN with the number of pipeline stages equal to the number of network stages (CBSN1) and CBSN with the number of pipeline stages equal to the number of network steps (CBSN2) for 16, 64, and 256 input records. The number of input records is limited to 2^{2n}, where RTHS can achieve the most optimal hardware implementation. It also suits the bitonic sorting network, with 2^n input format. However, the RTHS design supports various sizes for the input matrix. The number of CAS blocks inCBSN1 and CBSN2 is obtained from (1) and RTHS is obtained from the following equation:

$$\text{Num}_{\text{CAS}-\text{RTHS}} = \text{Num}_{\text{DPBN}}(D/4 \times \log_2 D(\log_2 D + 1)) \tag{10}$$

where Num_{DPBN} is the number of DPBNs, while D shows the number of inputs/outputs in a DPBN unit. According to Table II, the RTHS design needs 71.2% on average less CAS units compared to CBSN1. The dual-mode CAS blocks, used in RTHS, does not consume more resources compared to increasing/decreasing CAS blocks, because the XOR gate is implemented in the same LUT.

As we explained in the previous section, for a fair comparison, we assumed a pipeline stage between every two stages of DPBN in the bitonic network. CBSN2 is also considered in order to show the excessive increase in the number of its required registers. We put a pipeline stage between each two steps of a CBSN, highlighting the advantage of RTHS design in terms of resource consumption. This modification is seemed to be necessary, as CBSN2 occupies an excessive amount of registers when the number of inputs passes 32 records (tested for 64 and 128 records).

As we know, the maximum operating frequency depends on the minimum period of the sorting circuit clocks, which is limited to the largest critical path of the design. High resource requirements of CBSN1, for 64 input records and higher, nullifies the pipelining benefits to a certain extent, due to the increase in the number of steps. The impact of increasing pipeline stages directly affects the critical path and the minimum period of a clock cycle. Our minimalistic approach has allowed us to embed a pipeline stage between each step of DPBN units, resulting in 85.4% and 69.9% shorter critical path compared to CBSN1 in the best and the worst case, respectively. In response to an increase in the number of input records, we can simply add additional DPBN units to our design. Due to the parallel nature of our architecture, the critical path will remain unaffected, resulting in a predictable execution-time and better performance in real-time applications.

In Table II, the number of LUTs has dropped by 66.3% on average compared to CBSN1. Unlike LUTs, the number of registers is increased by 43.5% on average. As expected, the main reason behind this increase is the number of pipeline registers, used in the RTHS design.

Scalability is one of the most important issues in the sorting problem. According to our evaluations, the scalability of the RTHS design is far superior to CBSN. By increasing the number of inputs, we experience a little impact on the critical

TABLE II

COMPARISON OF RTHS DESIGN, CBSN WITH THE NUMBER OF PIPELINE STAGES EQUAL TO THE NUMBER OF NETWORK STAGES (CBSN1) AND CBSN WITH THE NUMBER OF PIPELINE STAGES EQUAL TO THE NUMBER OF NETWORK STEPS (CBSN2)

# of I/O	# of DPBN	# of I/O in DPBN	CAS blocks			Pipeline stages			Data width	LUTs			Registers			Critical path (ns)			Frequency (MHz)		
			CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS		CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS	CBSN 1	CBSN 2	RTHS
16	4	4	80	80	24	4	10	4	32-bit	*6,344 (2%)*	*7,058 (2.3%)*	*2,437 (0.8%)*	*1,536 (0.25%)*	*5,632 (0.93%)*	*2,051 (0.33%)*	*6.061*	*1.822*	*1.871*	*151.2*	*548.7*	*534.4*
									64-bit	*12,744 (4%)*	*14,098 (4.7%)*	*4,869 (1.6%)*	*3,072 (0.5%)*	*11,264 (1.8%)*	*4,099 (0.6%)*	*6.397*	*1.942*	*1.936*	*156.3*	*514.8*	*516.4*
64	8	8	672	672	192	6	21	6	32-bit	*53,376 (17%)*	*59,728 (20%)*	*17,413 (5%)*	*10,240 (1%)*	*45,060 (7.4%)*	*14,340 (2%)*	*9.138*	*1.923*	*1.886*	*109.4*	*520*	*530.2*
									64-bit	*107,136 (35%)*	*113,040 (37%)*	*34,823 (11%)*	*20,480 (3%)*	*90,112 (15%)*	*28,675 (4%)*	*10.27*	*2.025*	*2.013*	*97.3*	*493.8*	*496.7*
256	16	16	4,608	4,608	1,280	8	36	10	32-bit	*365,696 (120%)*	*379,340 (125%)*	*110,605 (36%)*	*57,344 (9%)*	*303,104 (50%)*	*90,115 (14%)*	*13.299*	*3.193*	*1.932*	*75.2*	*313.2*	*517.6*
									64-bit	*736,128 (242%)*	*758,339 (250%)*	*221,197 (72%)*	*114,688 (18%)*	*606,208 (99%)*	*180,227 (29%)*	*15.138*	*5.631*	*2.289*	*66*	*177.5*	*436.8*

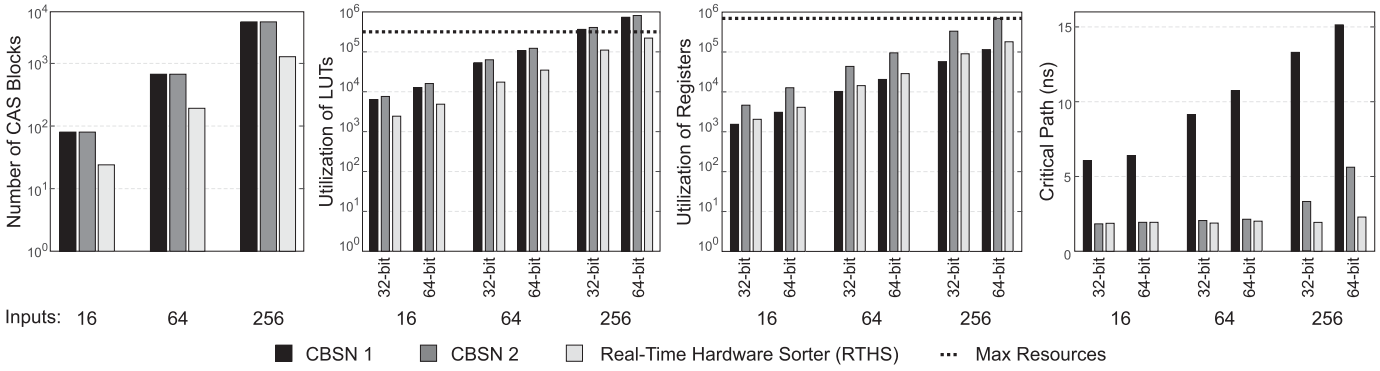

Fig. 10. Number of CAS blocks, LUTs, registers and critical path (ns) reported for the CBSN1, CBSN2, and the RTHS design.

path and a slow increase in resource requirements of our design, which greatly contribute to its scalability.

Table III and Fig. 11 put RTHS against PMT, SHMS, RSN, and the unary sorting for 16, 32, 64, and 256 input records. PMT and SHMS are not implemented for 256 input records due to the high complexity and resource requirements. Therefore, in these two cases, the results are estimated hypothetically. The unary design requires the smallest amount of resources for implementation. We use the 64-record format in RTHS to sort 32 input records. We put 32 incoming records in the top entries of the input matrix and assign the minimum value to the remaining records to be neutral in the sorting process. The top 32 records of the matrix will be sent to the output after the sorting process ends. Only some input registers and their corresponding sorting circuits will be removed from the 64 record design to further tailor the hardware for the 32-record use case. RSN is implemented, according to the pipelined design, presented in [31]. The results are obtained from the hardware part of the design, but the numbers will change if we factor in the presorting and merge-sort steps handled by software.

The number of LUTs used by RTHS has decreased by 85.8% and 87.3%, compared to PMT and SHMS designs respectively and has increased between 29.9% and 1.3× compared to the unary design. Like LUTs, we achieved 65.8% and 94.8% savings in the number of registers used, compared to PMT and SHMS designs, respectively.

The length of the critical path in RTHS design is far less than the other sorting methods and offers a relatively constant value. The critical path of RTHS relies on a single dual-mode CAS block, whereas in other sorting methods, it is equal to the maximum number of consecutive CAS blocks. If we consider the delay of the implicit switch to be constant, the total delay of RTHS will not rise by increasing the number of input records. However, in reality, the complexity of the switch leads to a more complex routing on the FPGA, which, in turn, increases the critical path gradually. The critical path of our design is shorter by 80.3%, 43.25%, and 26.5%, compared to

TABLE III
COMPARISON OF RTHS DESIGN AND UNARY, PMT, SHMS, AND RSN DESIGNS

# of I/O	Pipeline stages					Data width	LUTs					Registers					Critical path (ns)				
	Unary	PMT	SHMS	RSN	RTHS		Unary	PMT	SHMS	RSN	RTHS	Unary	PMT	SHMS	RSN	RTHS	Unary	PMT	SHMS	RSN	RTHS
16	4	4	15	4	4	32-bit	*1,875 (0.6%)*	*22,927 (7%)*	*18,665 (6.2%)*	*2,535 (0.8%)*	*2,437 (0.8%)*	*1,072 (0.17%)*	*8,140 (1.3%)*	*40,973 (6%)*	*2,570 0.4%)*	*2,051 (0.33%)*	*2.34*	*7.13*	*2.85*	*1.85*	*1.87*
						64-bit	*3,603 (1.2%)*	*47,001 (15%)*	*44,558 (14%)*	*5,034 (1.6%)*	*4,869 (1.6%)*	*2,096 (0.33%)*	*16,299 (2.7%)*	*90,141 (14%)*	*5,134 (0.9%)*	*4,099 (0.6%)*	*2.37*	*7.49*	*2.91*	*2.24*	*1.93*
32	5	5	31	4	6	32-bit	*4,094 (1.4%)*	*67,704 (22%)*	*76,493 (25%)*	*5,058 (1.7%)*	*16,753 (5.6%)*	*2,176 (0.35%)*	*23,973 (3.9%)*	*157,213 (27%)*	*5,140 (0.9%)*	*13,246 (2.2%)*	*2.55*	*9.83*	*3.10*	*2.24*	*1.88*
						64-bit	*7,806 (2.6%)*	*142,179 (46%)*	*175,936 (58%)*	*10,040 (3.3%)*	*33,015 (11%)*	*4,224 (0.7%)*	*45,445 (7.5%)*	*330,148 (54%)*	*10,268 (1.7%)*	*27,428 (4.5%)*	*2.59*	*10.1*	*3.21*	*2.52*	*1.97*
64	6	6	63	4	6	32-bit	*8,627 (2.8%)*	*223,635 (73%)*	*278,565 (92%)*	*10,079 (3.3%)*	*17,413 (5.7%)*	*4,416 (0.7%)*	*76,713 (12%)*	*606,394 (99%)*	*10,280 (1.7%)*	*14,340 (2.4%)*	*2.87*	*13.1*	*4.17*	*2.26*	*1.88*
						64-bit	*15,125 (5%)*	*429,380 (141%)*	*668,556 (220%)*	*20,002 (6.6%)*	*34,823 (11%)*	*8,512 (1.4%)*	*140,879 (23%)*	*1188532 (195%)*	*20,536 (3.4%)*	*28,675 (4.7%)*	*2.92*	*14.2*	*4.51*	*2.69*	*2.01*
256	8	-	-	4	10	32-bit	*37,631 (12%)*	-	-	*40,312 (13%)*	*110,605 (36%)*	*18,176 (2%)*	-	-	*41,120 (6.7%)*	*90,115 (14%)*	*3.49*	-	-	*2.43*	*1.93*
						64-bit	*56,964 (18%)*	-	-	*79,819 (26%)*	*221,197 (72%)*	*34,560 (5%)*	-	-	*82,144 (13.5%)*	*180,227 (29%)*	*3.38*	-	-	*2.98*	*2.29*

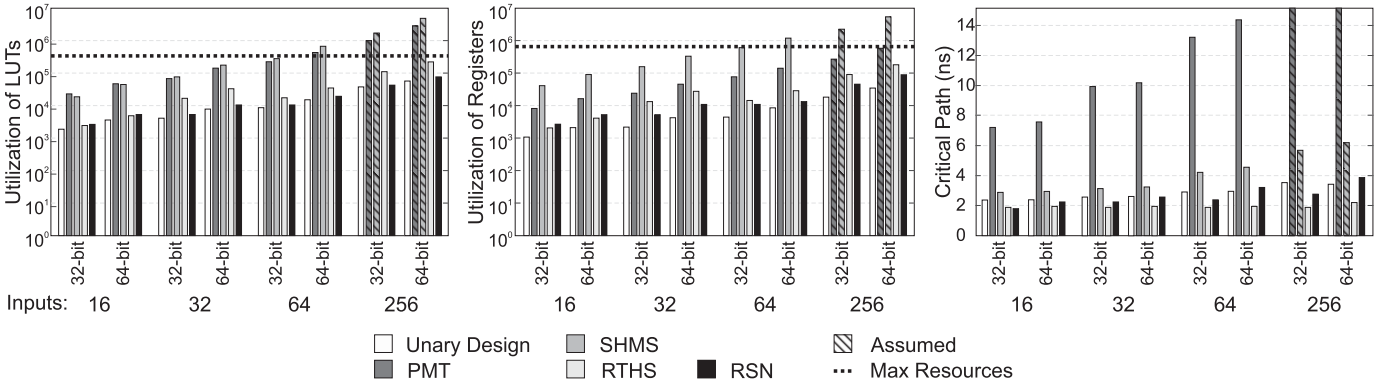

Fig. 11. Number of LUTs, registers, and critical path (ns) reported for the unary, PMT, SHMS, RSN, and RTHS designs.

the PMT, SHMS, and unary designs, respectively. Therefore, we have managed to achieve the highest frequency among other sorting designs.

C. Throughput Analysis

Throughput T for a sorter with N input records is calculated from the following equation:

$$T = N/t \times f \times \text{data}_{\text{width}} \tag{11}$$

where f is the frequency, t is the time required to sort N input records (in cycles) and $\text{data}_{\text{width}}$ is measured in bytes.

RTHS employs $N/2$ BRAMs and a BRAM control unit to store $\text{Num}_{\text{Pipeline stages}} \times \text{Num}_{I/O}$ incoming input records from the main memory. BRAMs are used to circumvent the delays of reading/writing records directly from the memory which further impact the throughput of the sorter. Virtex-7 benefits from dual-port BRAMs, resulting in simultaneous writes and reads. This number of records will certainly exceed the sorting capacity of RTHS. Hence, BRAMs transfer each set of input records to the sorting registers in two clock cycles serially. After the transfer is completed, the sorting procedure begins for each set of records. Using a software bitonic algorithm, the records will be fed to the sorter one sequence at a time. Each BRAM is divided into two sections. The second section is used to store the next sequence of incoming input records. So, by the time, sorter sorts out the records in the first

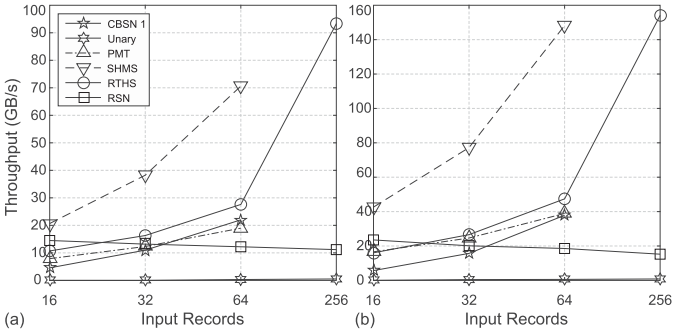

Fig. 12. Throughput of sorting methods for different number of input records. (a) 32-bit data width. (b) 64-bit data width.

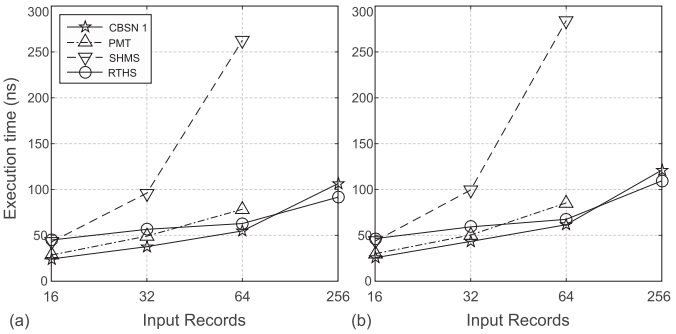

Fig. 13. Execution-time of sorting methods for different number of input records. (a) 32-bit data width. (b) 64-bit data width.

BRAM section, the upcoming sequence will be available in the second BRAM section to eliminate the memory conflict between writing the results and reading the next sequence of records. In the sorting procedure, each BRAM transfers sorted sequence of records, received in the previous sorting procedure, to the main memory and receives a new sequence of input records.

Fig. 12 compares the throughput of the RTHS design with the unary, PMT and SHMS solutions. The throughput for 256 input PMT and SHMS designs are absent because they cannot be implemented. The unary design takes 2^M cycles to sort N records with M-bit width, which is much longer than the CBSN delay (the longest critical path) [14]. If the data width is low, the unary design has proved to be cost-effective. Otherwise, it has the lowest throughput (close to zero) with 32-bit and 64-bit data widths.

Note that the RTHS design receives a completely unordered sequence of records in the beginning, but the PMT and SHMS methods receive sequences of partially sorted records as input and pass a single ordered sequence to their outputs. We have not considered the overhead of preliminary partial sorting operations in our comparison, regarding the aforementioned methods.

D. Execution Time Analysis

Fig. 13 shows the execution-time of sorting methods for a different number of input records. At the beginning of time, only one sequence of records will be assigned to sorters. Their execution times will be the difference between the time that records were fed to the sorters and the time that sorted records

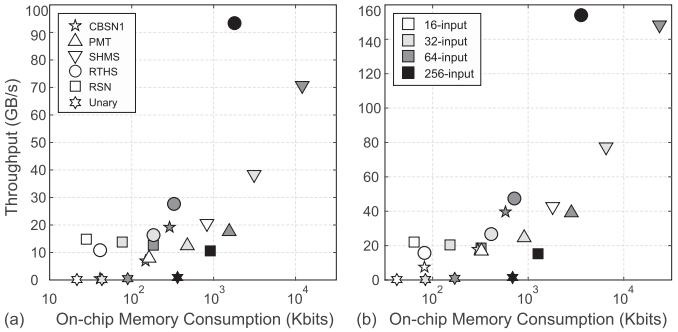

Fig. 14. Memory-efficiency between five sorting methods for different number of input records. (a) 32-bit data width. (b) 64-bit data width.

are received at their output. As expected, the execution time of RTHS proved to be more stable than other methods. Note that the pipeline technique does not affect execution time in Fig. 13, because we only passed one sequence of input records to the sorters. The serialized process of the SHMS method suppresses its performance severely. The unary method posted prolonged delays in our test scenario to the point that made it irrelevant to include its scores in Fig. 13. The execution time of the RSN method is not included in Fig. 13 since the execution time of RSN depends on the software sorter and is greater than other methods.

E. Memory Efficiency Analysis

Fig. 14 compares the memory efficiency between five sorting methods. The ideal sorter should achieve the maximum throughput with the least amount of memory required. The SHMS method that offers more throughput than RTHS is located on the right side of the plot because it consumes a lot of memory. The RTHS method for 256 records with 64-bit width offers the optimized memory usage on the top left of the memory- efficiency plot and has the best performance.

F. Power Consumption Analysis

Power consumption is a substantial aspect of every digital system design. For FPGA chips, the power consumption depends on the design and operating conditions, the target device and its operating frequency. Furthermore, power consumption is heavily dependent on resource utilization, placement and route, mapping, and logic partition [33]. The power consumption of an FPGA chip is derived from the total of static and dynamic power consumptions, which include clock signal power, design signals, logic blocks, and I/O. Fig. 15 shows the power consumption of the proposed method in comparison with other sorting methods. As expected, increasing the number of input records will increase the power consumption of all methods, present in Fig. 15. The unary method offers the lowest power consumption, due to its drastic reduction of required resources. On the contrary, the SHMS method demonstrates the highest power consumption, considering its high resource requirements and operating frequency. The RTHS method needs more power than the CBSN1 and unary methods. High operating frequency and total system dependence on a controller have negatively impacted the

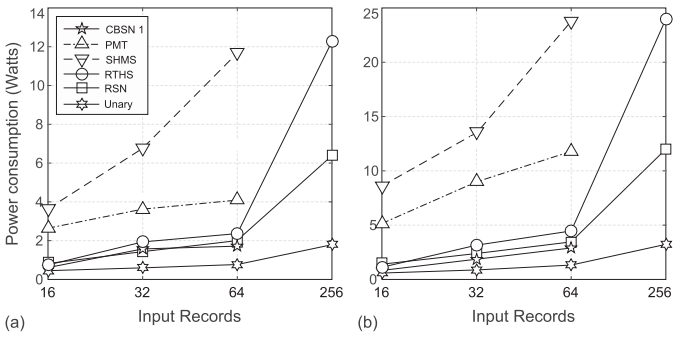

Fig. 15. Power consumption of sorting methods for different number of input records. (a) 32-bit data width. (b) 64-bit data width.

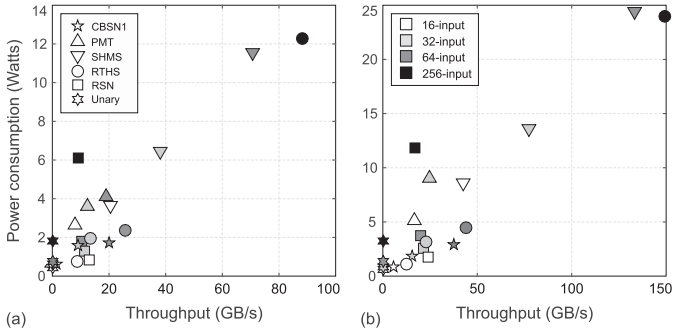

Fig. 16. Throughput to power consumption ratio of five sorting methods. (a) 32-bit data width. (b) 64-bit data width.

power consumption of the proposed method, compared to the CBSN1, despite using less LUTs. The RTHS method still requires less power than the PMT and SHMS methods, due to the drastic decrease in resource consumption. Despite the fact that low power consumption is always desirable, the throughput to power consumption ratio provides a more equitable insight for comparing different sorting circuits. Fig. 16 compares RTHS against other sorting methods, based on the throughput to power ratio. The acceptable results are the ones, placed under the $y = x$-axis. In other words, the right bottom of the graph represents the most ideal design, which translates to the most efficiency coupled with the lowest power consumption. The proposed RTHS method is always below the $y = x$-axis except for the implementation of 32 input records with 32-bit data width.

VII. Conclusion

Sorting is one of the most important tasks, employed in many applications. FPGAs provide high-throughput and memory-efficient solutions to implement parallel and pipeline designs of various architectures. The bitonic sorting network offers high throughput, but it lacks memory-efficiency and requires a lot of resources to be implemented. In this paper, we have introduced a novel MDSA and high-performance hardware sorter. It only takes six phases for RTHS to sort any number of input records. We have presented a detailed analysis of RTHS, in terms of resource utilization, execution-time, memory, and throughput. RTHS reduces the required resources considerably. The number of LUTs is reduced by 66.3%, compared to the CBSN. We also managed to outdo the state-of-the-art method SHMS and decrease the number of LUTs and registers by 87.3% and 94.8%, respectively. In the future, we plan to implement big Min/Max queues for task scheduling in real-time systems on multicore platforms.

References

[1] U. Güvenç and F. Katircioğlu, "Escape velocity: A new operator for gravitational search algorithm," *Neural Comput. Appl.*, vol. 31, no. 1, pp. 27–42, Apr. 2017. doi: 10.1007/s00521-017-2977-9.

[2] K. Sujatha, P. V. N. Rao, A. A. Rao, V. G. Sastry, V. Praneeta, and R. K. Bharat, "Multicore parallel processing concepts for effective sorting and searching," in *Proc. Int. Conf. Signal Process. Commun. Eng. Syst.*, Jan. 2015, pp. 162–166.

[3] G. Graefe, "Implementing sorting in database systems," *ACM Comput. Surv.*, vol. 38, no. 3, p. 10, Sep. 2006. [Online]. Available: http://doi.acm.org/10.1145/1132960.1132964

[4] S. W. Al-Haj Baddar and B. A. Mahafzah, "Bitonic sort on a chained-cubic tree interconnection network," *J. Parallel Distrib. Comput.*, vol. 74, no. 1, pp. 1744–1761, Jan. 2014. doi: 10.1016/j.jpdc.2013.09.008.

[5] M. Kik, "Merging and merge-sort in a single hop radio network," in *SOFSEM 2006: Theory and Practice of Computer Science*, J. Wiedermann, G. Tel, J. Pokorný, M. Bieliková, and J. Štuller, Eds. Berlin, Germany: Springer, 2006, pp. 341–349.

[6] Y. Zhang, L. Li, M. Ripperger, J. Nicho, M. Veeraraghavan, and A. Fumagalli, "Gilbreth: A conveyor-belt based pick-and-sort industrial robotics application," in *Proc. 2nd IEEE Int. Conf. Robot. Comput. (IRC)*, Jan. 2018, pp. 17–24.

[7] D. C. Stephens, J. C. R. Bennett, and H. Zhang, "Implementing scheduling algorithms in high-speed networks," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 6, pp. 1145–1158, Jun. 1999.

[8] Y. Tang and N. W. Bergmann, "A hardware scheduler based on task queues for FPGA-based embedded real-time systems," *IEEE Trans. Comput.*, vol. 64, no. 5, pp. 1254–1267, May 2015.

[9] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-cost sorting network circuits using unary processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 8, pp. 1471–1480, Aug. 2018.

[10] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Digit. Equip. Corp., Maynard, MA, USA, Tech. Rep. 124, 1994.

[11] A. Rasmussen *et al.*, "Tritonsort: A balanced and energy-efficient large-scale sorting system," *ACM Trans. Comput. Syst.*, vol. 31, no. 1, Feb. 2013, Art. no. 3. [Online]. Available: http://doi.acm.org/10.1145/2427631.2427634

[12] S.-W. Jun, S. Xu, and A. Arvind, "Terabyte sort on FPGA-accelerated flash storage," in *Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2017, pp. 17–24.

[13] A. Stillmaker, L. Stillmaker, and B. Baas, "Fine-grained energy-efficient sorting on a many-core processor array," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst.*, Dec. 2012, pp. 652–659.

[14] B. Jan, B. Montrucchio, C. Ragusa, F. G. Khan, and O. Khan, "Fast parallel sorting algorithms on GPUs," *Int. J. Distrib. Parallel Syst.*, vol. 3, no. 6, p. 107, Nov. 2012.

[15] A. Srivastava, R. Chen, V. K. Prasanna, and C. Chelmis, "A hybrid design for high performance large-scale sorting on FPGA," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2015, pp. 1–6.

[16] W. Song, D. Koch, M. Luján, and J. Garside, "Parallel hardware merge sorter," in *Proc. IEEE 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2016, pp. 95–102.

[17] S.-H. Lin, P.-Y. Chen, and Y.-N. Lin, "Hardware design of low-power high-throughput sorting unit," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1383–1395, Aug. 2017.

[18] S. Mashimo, T. V. Chu, and K. Kise, "High-performance hardware merge sorter," in *Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2017, pp. 1–8.

[19] M. He, X. Wu, S. Q. Zheng, and B. Englert, "Optimal sorting algorithms for a simplified 2D array with reconfigurable pipelined bus system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 3, pp. 303–312, Mar. 2010.

[20] R. Chen and V. K. Prasanna, "Computer generation of high throughput and memory efficient sorting designs on FPGA," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3100–3113, Nov. 2017.

[21] P. P. Puschner and A. V. Schedl, "Computing maximum task execution times—A graph-based approach," *Real-time Syst.*, vol. 13, no. 1, pp. 67–91, Jul. 1997. doi: 10.1023/A:1007905003094.

[22] L. Kohútka and V. Stopjaková, "Rocket queue: New data sorting architecture for real-time systems," in *Proc. IEEE 20th Int. Symp. Design Diagnostics Electron. Circuits Syst. (DDECS)*, Apr. 2017, pp. 207–212.
[23] A. Rjabov, "Hardware-based systems for partial sorting of streaming data," in *Proc. 15th Biennial Baltic Electron. Conf. (BEC)*, Oct. 2016, pp. 59–62.
[24] A. D. G. Biroli and J. C. Wang, "A fast architecture for finding maximum (or minimum) values in a set," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2014, pp. 7565–7569.
[25] Z. K. Baker and V. K. Prasanna, "An architecture for efficient hardware data mining using reconfigurable computing systems," in *Proc. 14th Annu. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 2006, pp. 67–75.
[26] K. E. Batcher, "Sorting networks and their applications," in *Proc. Spring Joint Comput. Conf. (AFIPS)*, Apr. 1968, pp. 307–314. [Online]. Available: http://doi.acm.org/10.1145/1468075.1468121
[27] C. Layer, D. Schaupp, and H.-J. Pfleiderer, "Area and throughput aware comparator networks optimization for parallel data processing on FPGA," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 405–408.
[28] J. Angermeier, E. Sibirko, R. Wanka, and J. Teich, "Bitonic sorting on dynamically reconfigurable architectures," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 314–317.
[29] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, Oct. 2011, Art. no. 35. [Online]. Available: http://doi.acm.org/10.1145/1978802.1978814
[30] M. Ricco, L. Mathe, and R. Teodorescu, "FPGA-based implementation of sorting networks in MMC applications," in *Proc. 18th Eur. Conf. Power Electron. Appl. (EPE ECCE Europe)*, Sep. 2016, pp. 1–10.
[31] V. Sklyarov and I. Skliarova, "High-performance implementation of regular and easily scalable sorting networks on an FPGA," *Microprocessors Microsyst.*, vol. 38, no. 5, Jul. 2014, pp. 470–484.
[32] *Vivado Design Suite User Guide: Design Flows Overview*. Accessed: Oct. 20, 2018. [Online]. Available: http://www.xilinx.com/support/documentation/
[33] H. G. Lee, S. Nam, and N. Chang, "Cycle-accurate energy measurement and high-level energy characterization of FPGAS," in *Proc. 4th Int. Symp. Qual. Electron. Design*, Mar. 2003, pp. 267–272.

Amin Norollah received the B.S. degree in computer engineering from Islamic Azad University, South Tehran Branch, Tehran, Iran, in 2015. He is currently working toward the M.Sc. degree at the Department of Computer Engineering, Iran University of Science and Technology, Tehran.

His current research interests include computer architecture, hardware accelerators, real-time systems, and reconfigurable computing.

Danesh Derafshi received B.E. degree in hardware engineering from Shiraz University, Shiraz, Iran, in 2016. He is currently working toward the M.Sc. degree at Iran University of Science and Technology, Tehran, Iran.

His current research interests include embedded real-time systems and reconfigurable architectures.

Hakem Beitollahi received the B.S. degree in computer engineering from the University of Tehran, Tehran, Iran, in 2002, the M.S. degree from the Sharif University of Technology, Tehran, in 2005, and the Ph.D. degree from Katholieke Universiteit Leuven, Leuven, Belgium, in 2012.

He is currently an Assistant Professor and the Head of the Hardware and Computer Architecture Branch, Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran. He and his graduate students are investigating new architectures, integration techniques, and systems software techniques for reconfigurable computing and real-time systems. His current research interests include real-time systems, fault tolerance and dependability, reconfigurable computing, and hardware accelerators.

Mahdi Fazeli received the M.Sc. and Ph.D. degrees in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2005 and 2011, respectively.

Since 2011, he has been at the Department of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran, where he is currently an Associate Professor. He was the Founder of the Dependable Systems and Architectures Laboratory, IUST, where he has been the Chair since 2012. He has authored or coauthored more than 40 papers in reputable journals and conferences. His current research interests include system-level power analysis and management, real-time systems, low-power circuits and systems, reconfigurable computing, and reliability modeling and evaluation.