MODULAR DESIGN OF HIGH-THROUGHPUT, LOW-LATENCY SORTING UNITS

by

Amin Farmahini-Farahani

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Master of Science

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2012

To my mother and father

ACKNOWLEDGMENTS

I have been lucky to have the possibility of working with two advisors. My advisors are Professor Michael Schulte and Professor Katherine Compton, who have always been most generous with their time and encouragement. I wish to express my greatest thanks to them to give me the opportunity of learning from two different perspectives.

I thank Professor Wesley Smith and Engineer Thomas Gorski for giving me a chance to work on intriguing projects for the Large Hadron Collider and letting me have the privilege of living in the high-energy physics world. I would like to extend my gratitude to Robert Fobes for technical, cultural, and amusing things he taught me.

I also thank my fellow graduate students at the Madison Embedded Systems and Architecture (MESA) and Predictive High-Performance Architecture Research Mavens (PHARM) labs for creating and sharing an excellent group atmosphere. I have made many friends in the past four years. Jake Adriaens, Paula Aguilera, Ben Buchli, Daniel Chang, Henry Duwe, Philip Garcia, Syed Zohaib Gilani, Tony Gregerson, Mitch Hayenga, Jung-Seob Lee, Steve Naumov, Andrew Nere, David Palframan, Kyle Rupnow, Charles Tsen, and Hsiang-Kuo Tang are a few, but there are more than I can name.

I thank my fellow Iranian friends for sharing this experience with me, particularly Hamid Reza Ghasemi, Peiman Hematti, Shayda Malekpour, Shirin Malekpour, Shirzad Malekpour, Somayeh Sardashti, and Arsham Shahlari.

Finally, I would like to express my deep gratitude to my mother, father, and brother for supporting me. Their encouragement and love were the great power for me to overcome all troubles I faced.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

High-throughput and low-latency sorting is a key requirement in many applications that deal with large amounts of data. Searching and high-energy physics systems require a considerable number of sorting units. The particle detectors in CERN's Large Hadron Collider require hundreds of fast sorting units. To provide the performance and flexibility needed in high-energy physics experiments, these sorting units are often implemented using high-end FPGA devices. This thesis presents efficient techniques for designing high-throughput, low-latency sorting units. Our sorting architectures utilize modular design techniques that hierarchically construct large sorting units from smaller building blocks. The sorting units are optimized for situations in which only the M largest numbers from N inputs are needed, since this situation commonly occurs in many applications for scientific computing, data mining, network processing, digital signal processing, and high-energy physics. We utilize our proposed techniques to design parameterized, pipelined, and modular sorting units. A detailed analysis of these sorting units indicates that as the number of inputs increases their resource requirements scale linearly, their latencies scale logarithmically, and their frequencies remain almost constant. When synthesized to a 65-nm TSMC technology, a single pipelined 256-to-4 sorting unit with 19 stages can perform more than 2.7 billion sorts per second with a latency of about 7 ns per sort. When implemented on a Virtex-5 FPGA, the same sorting unit can perform roughly 200 million sorts per second with a latency of about 95 ns per sort. We also propose iterative sorting techniques, in which a small sorting unit is used several times to find the largest values.

1 INTRODUCTION

Sorting is an important operation in a wide range of applications including data mining, databases [7, 19, 31], digital signal processing [47, 48], network processing, communication switching systems [4, 58], scientific computing [15], searching, scheduling [51], pattern recognition, robotics [10], image and video processing [11, 12, 17, 49], and high-energy physics (HEP) [23, 55]. For applications that require very high-speed sorting, hardware sorting units are often implemented using either ASICs or FPGAs to meet performance requirements [13, 28, 31, 33, 38, 41, 49]. Based on target applications, hardware sorting units vary greatly not only in architecture but also in the number of inputs and the width of inputs that they can process. For instance, only 9 to 25 inputs need to be processed in certain filters [11, 12], while the number of inputs can vary from 25 to 81 (or even higher) in certain image processing applications [45]. High-speed sorters on FPGAs in HEP applications deal with 128 to 256 data samples in 100 ns processing cycles [18, 23]. Thousands of inputs are sorted in video [49] and database applications [20, 31]. In general, inputs can be b-bit integers ($8 \leqslant \mathsf{b} \leqslant 64$), floating-point numbers, or even compressed data values.

Most previous research on sorting units has focused on the situation in which the sorting unit must produce all of its inputs in sorted (increasing or decreasing) order. In many applications, however, only the M largest (or smallest) output values need to be selected from a total of N input values, where $\mathsf{M} < \mathsf{N}$. For example, in many HEP applications, only the M most energetic particles may be of interest. Similarly, in signal processing applications, only the M strongest signals or M closest points in space may need to be analyzed. In data mining, searching, and database systems, only top query outputs that score the most with respect to a given search key may need further processing. Furthermore, depending on the application, the M largest (smallest) outputs may not need to be in order. We refer to units

that only return the M largest (smallest) outputs, but do not guarantee that these M outputs are sorted, as max(min)-set-selection units. We refer to units that only return the M largest (smallest) outputs in sorted order as partial sorting units.

This thesis focuses on the design of partial sorting and max-set-selection units that return the $M = 2^m$ largest values from $N = 2^n$ inputs, where m and n are each whole numbers and $1 \leqslant M < N$[1]. We refer to these units as N-to-M partial sorting and max-set-selection units. Our units discard small inputs as early as possible to reduce the sorting units' latency and hardware complexity. We investigate the design and VLSI implementations of partial sorting and max-set-selection units with low latency, high throughput, and modest resource requirements. Our designs are based on Batcher's bitonic and odd-even merge sorting networks [8, 21], which are widely used in VLSI and FPGA implementations due to their simplicity, regularity, and parallelism. The proposed units are scalable in terms of both the number of inputs and the number of outputs. We also present a generalized platform-independent methodology for designing high-performance pipelined partial sorting and max-set-selection units for which the width of the data to be sorted and the pipeline depth can easily be varied.

This research is an extension of our previous work on FPGA-based sorting units in the Large Hadron Collider (LHC) [18]. The main contributions of this dissertation and [18] are:

- Modular techniques for designing N-to-M partial sorting and max-set-selection units. The units are composed of small and regular building blocks connected in a modular fashion, thereby reducing verification time and simplifying the design process. Our designs have low latency, high throughput, and modest resource requirements, can

[1]Straightforward modifications to our designs allow the M smallest values, rather than the M largest values, to be output. It is also feasible to remove the current restriction that M and N are integer powers of two using techniques similar to those presented in [4, 25, 32, 37, 40].

be pipelined easily, have parameterized pipeline depth and data widths, and scale well to large values of N and M. Moreover, our techniques are independent of the bit-width and type of input values.

- A detailed analysis of our proposed partial sorting and max-set-selection units that includes both theoretical and synthesis estimates of our units' latency, throughput, and resource requirements. This analysis indicates that for a given number of outputs, resource requirements scale linearly with the number of inputs, latency scales logarithmically with the number of inputs, and the frequency remains nearly constant. Compared to conventional sorting units, which return all of their inputs in sorted order, our N-to-M partial sorting and max-set-selection units have much lower latency and area.

- A discussion of how the proposed max-set-selection units may be utilized iteratively to find the largest values from a set of data. This approach may lower resource requirements, storage cost, and I/O requirements at the cost of increased latency and decreased throughput.

To the best of our knowledge, this is the first time that N-to-M partial sorting networks have been presented and analyzed. In this work, we propose fast parallel sorting algorithms for finding/sorting the M largest values from N inputs and then design scalable architectures based on the proposed algorithms. Our N-to-M partial sorting networks have lower latency than any previous sorting designs when producing only the M largest values. Furthermore, our N-to-M max-set-selection units further decrease the latency and resource requirements by not producing their outputs in sorted order. Our parallel units target applications that require very low-latency sorting. Our iterative units target applications that require moderate-latency sorting by trading increased latency for reduced area and I/O bandwidth. Although our sorting units were originally designed for HEP experiments in the Large Hadron Collider, our methodology can be

utilized to design high-speed sorting and max-set selection units for a wide range of applications.

The remainder of this dissertation is organized as follows. Chapter 2 describes previous sorting networks and provides background information for our work. Chapter 3 presents our new partial sorting and max-set-selection units. Chapter 4 gives synthesis results for our proposed units. Chapter 5 shows how these units are used iteratively to sort data. Chapter 6 discusses related research on sorting algorithms and architectures. Chapter 7 concludes the dissertation.

2 PARALLEL SORTING NETWORKS

A sorting network is a collection of interconnected compare-and-exchange (CAE) blocks that guides a parallel set of inputs to a parallel set of outputs in sorted order. Each CAE block has two inputs and two outputs. If the input values are already in order, they are directed to the corresponding outputs; otherwise, they are swapped.

There are two types of CAE blocks, called increasing and decreasing CAE blocks, used in hardware-based sorting units. Fig. 2.1 shows the high-level implementations (left) and schematic symbols (right) for three building blocks used in previous sorting units and in our designs. Fig. 2.1(a) shows an increasing CAE block, which outputs its two inputs in ascending order. A decreasing CAE block, shown in Fig. 2.1(b), outputs its inputs in descending order. Decreasing and increasing CAE blocks are identical, except for their wiring. Each CAE block contains a comparator and two multiplexers. We also define Max units which are used in our designs. A Max unit, shown in Fig. 1(c), takes two inputs and returns the larger input. Note that the $\oplus$ and $\ominus$ symbols determine the type of the block in Fig. 2.1.

A sorting network usually consists of a series of stages in which each stage contains a number of CAE blocks that operate in parallel. The latency of a sorting network is proportional to its depth (the number of consecutive CAE blocks). Two popular parallel sorting networks that currently have the lowest known latency for hardware implementation are the bitonic and odd-even merge sorting networks proposed by Batcher [8]. The structure of a sorting network is fixed, regardless of the value of the numbers being sorted and the results of the comparisons. Sorting networks are a common solution for hardware-based sorting. Their parallel nature allows them to perform sorting much faster than the $\mathsf{O}(\mathsf{N} \times \log(\mathsf{N}))$ time achievable by the fastest sequential software-based sorting algorithms. A sorting network may also be pipelined to further increase throughput.

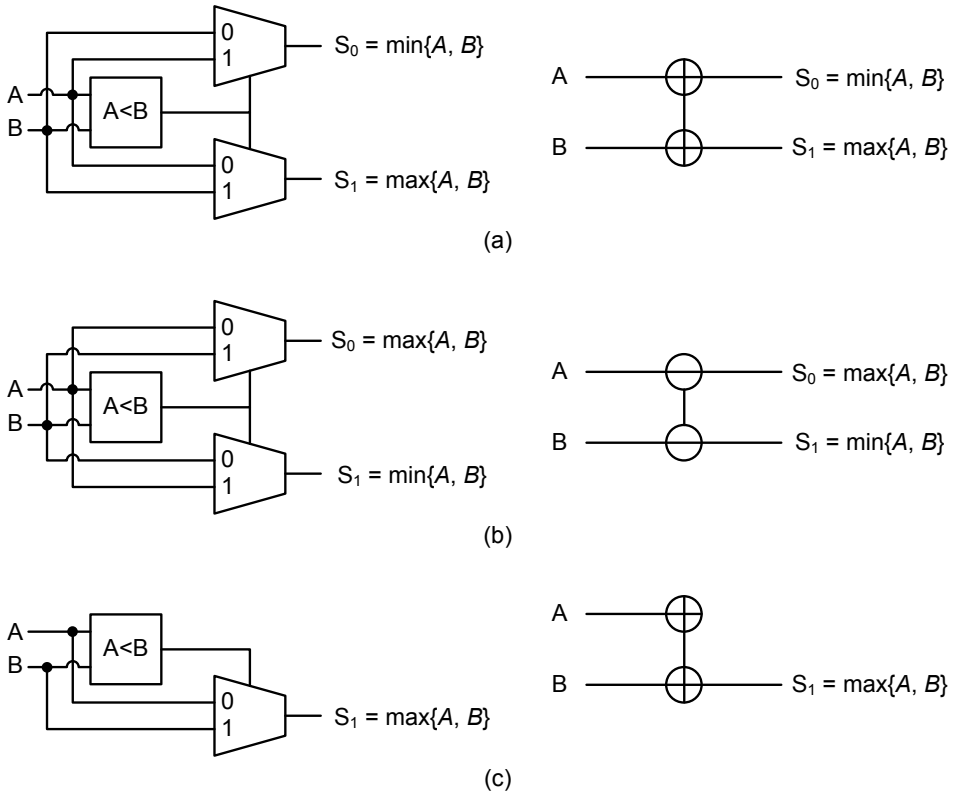

Figure 2.1: The high-level implementation (left) and schematic symbol (right) of building blocks for sorting networks.

2.1 Bitonic Sorting Networks

Concatenating an ascending and a descending sequence forms a single bitonic sequence. A bitonic sorting network recursively merges an ascending and a descending sequences each of length N/2 to make a sorted sequence of length N [8]. Each bitonic sorting network is composed of a number of bitonic merging units to merge bitonic sequences.

A K-input bitonic merging unit (denoted as BM-K) contains $\log_2(K)$ stages of parallel CAE blocks, where each stage corresponds to a CAE stage with K/2 CAE blocks. Therefore, a BM-K requires $\log_2(K) \times K/2$

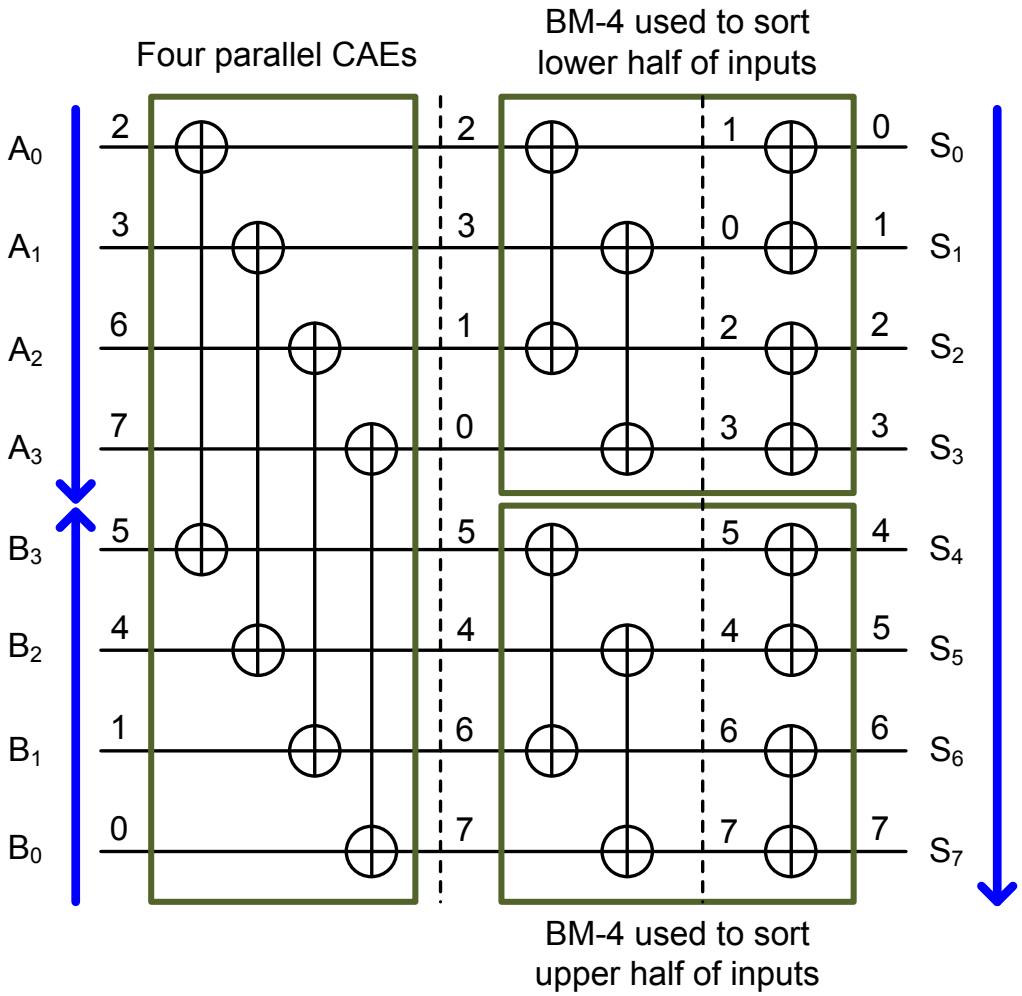

Figure 2.2: An increasing 8-input bitonic merging unit ($\oplus$BM-8) that is composed of four parallel CAE blocks followed by two parallel BM-4 units. The bitonic input sequence $\{2, 3, 6, 7, 5, 4, 1, 0\}$ is the concatenation of the increasing sequence $\{2, 3, 6, 7\}$ and the decreasing sequence $\{5, 4, 1, 0\}$.

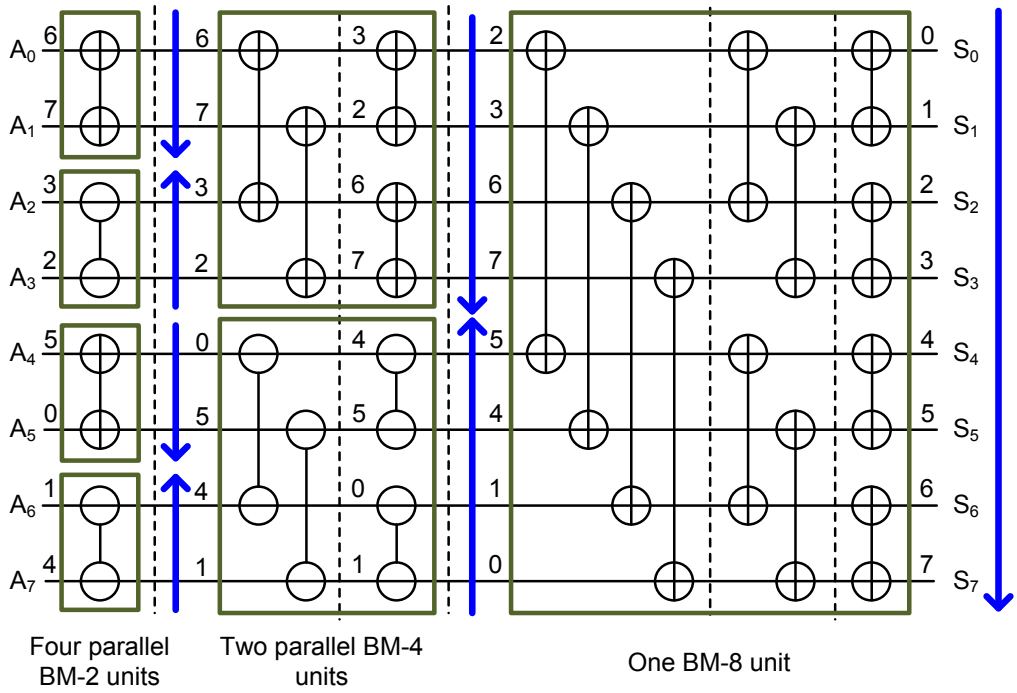

Figure 2.3: The CAE network for an 8-input bitonic sorting unit with six CAE stages and 24 CAE blocks, made up of increasing ($\oplus$) and decreasing ($\ominus$) bitonic merging units. Arrows point in the direction of increasing values.

CAE blocks. For instance, an increasing BM-8 (denoted as $\oplus$BM-8) unit has three CAE stages and requires $3 \times (8/2) = 12$ CAE blocks, as shown in Fig. 2.2. In this figure, the arrows point in the direction of increasing numbers and the dashed lines separate CAE stages. A BM-K unit is itself composed of a level of K/2 parallel CAE blocks followed by two parallel BM-(K/2) units. The $\oplus$BM-8 is constructed from a level of four parallel CAE blocks followed by two parallel BM-4 units that have four CAE blocks each. A decreasing BM-8 (denoted as $\ominus$BM-8) has a similar structure to an increasing BM-8, but it is only constructed from decreasing CAE blocks and the sorted outputs are in descending order.

An 8-input bitonic sorting unit has four parallel BM-2 units, two parallel

BM-4 units, and one BM-8 unit, as shown in Fig. 2.3. Thus, this sorting unit has $1 + 2 + 3 = 6$ CAE stages. Assuming the unit is pipelined so each stage takes one clock cycle, it can generate the sorted outputs in 6 cycles and can begin a new sort each cycle. In an N-input bitonic sorting unit, there are equal numbers of increasing and decreasing BM units in each level, excluding the last level, which has only either an increasing BM-N unit or a decreasing BM-N unit. In general, an N-input bitonic sorting unit is composed of N/2 BM-2 units, N/4 BM-4 units, N/8 BM-8 units, ..., two BM-N/2 units, and one BM-N unit. In this design, BM-K units are followed by BM-(2K) unit(s), where $K = 2, 4, 8, ..., N/2, N$, $2 \leqslant K \leqslant N$ and K and N are integer powers of 2. Therefore, since an N-input bitonic sorting unit has $\log_2(N)$ consecutive BM-K units, where each BM-K unit has $\log_2(K)$ CAE stages, an N-input bitonic sorting unit has $\log_2(N) \times (\log_2(N) + 1)/2$ CAE stages. Since each stage has N/2 CAE blocks, the total number of CAE blocks in an N-input bitonic sorting unit is $N \times \log_2(N) \times (\log_2(N) + 1)/4$. For example, 16-input, 32-input, and 256-input bitonic sorting units require 10, 15, and 36 CAE stages and have 80, 240, and 4,608 CAE blocks, respectively.

2.2 Odd-even Merge Sorting Networks

An odd-even merge sorting network recursively merges two ascending sequences of length K/2 to make a sorted sequence of length K. Each odd-even merge sorting network is composed of a number of odd-even merging units. A K-input odd-even merging unit (OEM-K) merges two ascending input sequences into a single ascending output sequence. It contains $\log_2(K)$ CAE stages, where each stage has between K/4 and K/2 CAE blocks. An OEM-K takes two length K/2 ascending sequences, A and B. The OEM-K merges the input values having odd indices in A with the input values having odd indices in B, and also merges input values in A and B having even indices. The result is a sorted sequence of values with odd indices (S_O) and

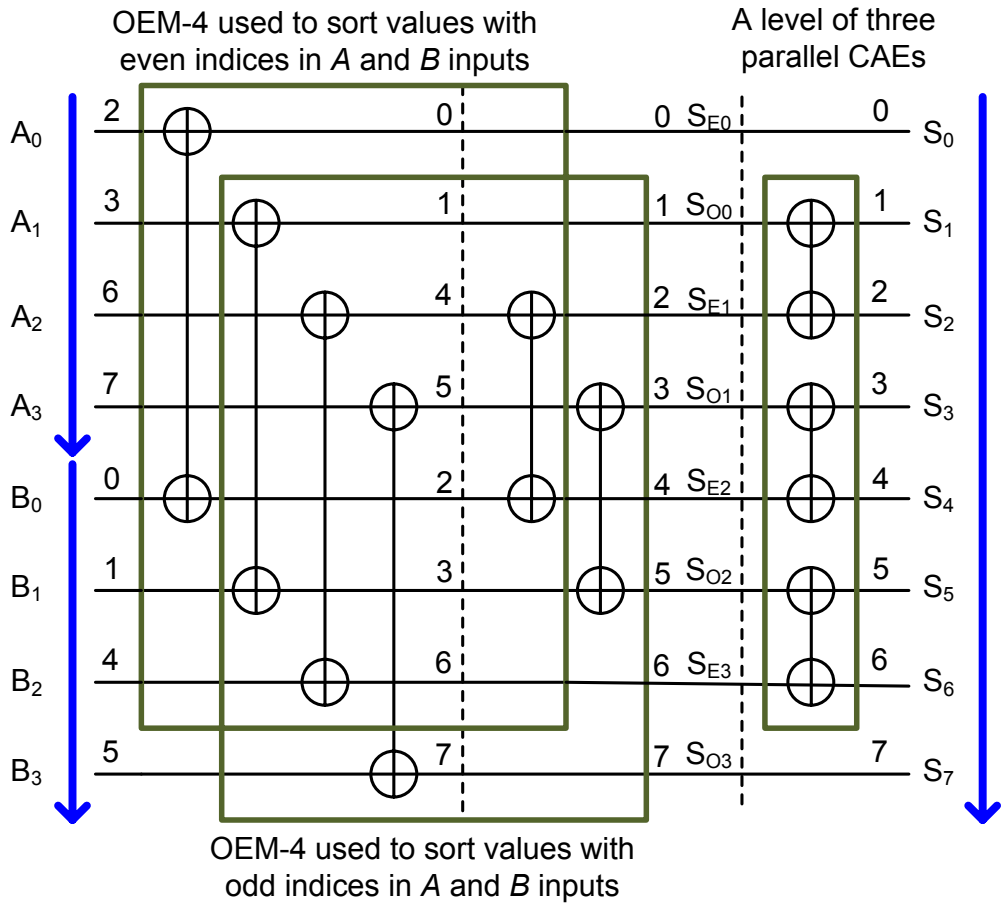

Figure 2.4: An 8-input odd-even merge unit (OEM-8) that is composed of two OEM-4 units and a level of three parallel CAE blocks.

a sorted sequence of values with even indices (S_E). S_O and S_E are generated recursively, separately, and in parallel. In the final stage, the S_O and S_E sequences are merged to generate a sorted sequence with K values, S_0 to S_{K-1}. The merging process is simply a compare-and-exchange of values in the S_O and S_E sequences.

An 8-input odd-even merging unit is shown in Fig. 2.4. An OEM-K unit is composed of two parallel OEM-K/2 units, followed by a level of $(K/2-1)$ parallel CAE blocks. Unlike BM units, OEM units are only built from

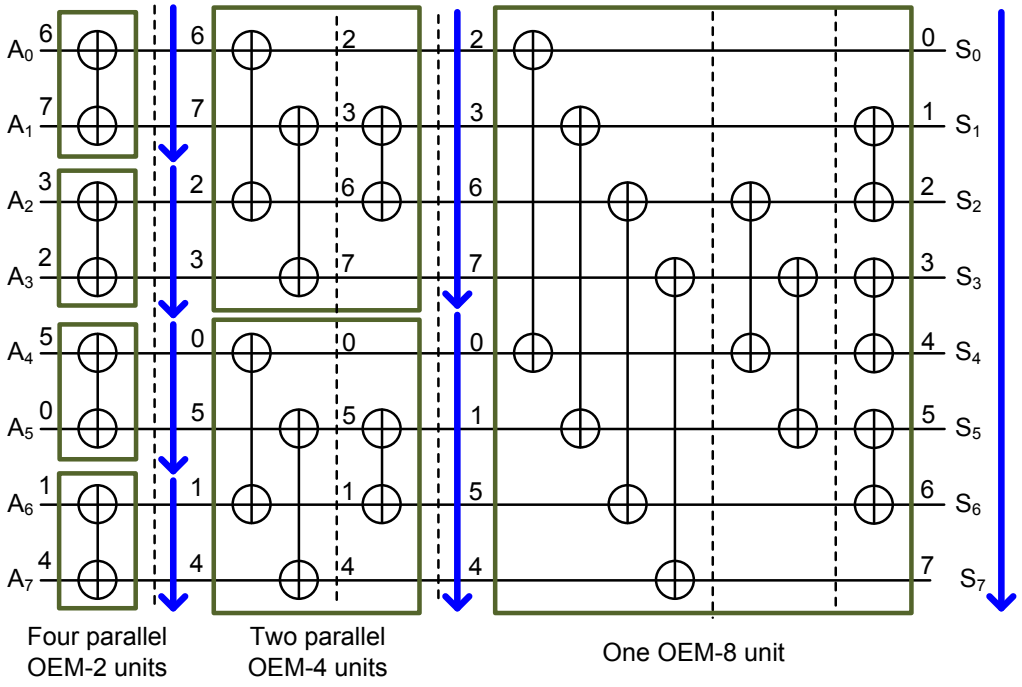

Figure 2.5: The CAE network for an 8-input odd-even merge sorting unit with six CAE stages and 19 CAE blocks.

increasing CAE blocks. The total number of CAE blocks for an OEM-K unit is $(K/2) \times (\log_2(K) - 1) + 1 = \log_2(K/2) \times (K/2) + 1$. Thus, an OEM-8 unit has three CAE stages and $2 \times 4 + 1 = 9$ CAE blocks. It is constructed from two parallel OEM-4 units that each have three parallel CAE blocks followed by a level of three parallel CAE blocks.

An 8-input odd-even merge sorting unit has four OEM-2 units, two OEM-4 units, and one OEM-8 unit, as shown in Fig. 2.5. It requires four parallel OEM-2 units, two parallel OEM-4 units, and a single OEM-8 unit, leading to a sorting unit with $1 + 2 + 3 = 6$ CAE stages. Assuming the unit is pipelined with one stage per clock cycle, it can generate the sorted outputs in six cycles. In general, an N-input odd-even merge sorting unit is

Table 2.1: The required number of CAE blocks and CAE stages for 2^n-input bitonic and odd-even merge sorting units

# of inputs and outputs ($N=M=2^n$)	# of CAE Stages	# of CAE blocks		
		Bitonic	Odd-even	Difference
8	6	24	19	5
16	10	80	63	17
32	15	240	191	49
64	21	672	543	129
128	28	1,792	1,471	321
256	36	4,608	3,839	769

composed of $N/2$ OEM-2 units[1], $N/4$ OEM-4 units, $N/8$ OEM-8 units, ..., two OEM-$N/2$ units, and one OEM-N unit. In this design, OEM-K units are followed by OEM-$2K$ unit(s), where $K = 2, 4, 8, ..., N/2, N$, and K and N are integers power of 2. Therefore, since an N-input odd-even merge sorting unit is composed of $\log_2(N)$ consecutive OEM-K units, where each OEM-K unit has $\log_2(K)$ CAE stages, an N-input odd-even merge sorting unit has $(\log_2(N)) \times (\log_2(N) + 1)$ stages. This is equivalent to the number of CAE stages in an N-input bitonic sorting unit. In addition, since an OEM-K has $\log_2(K/2) \times (K/2) + 1$ CAE blocks, an N-input odd-even merge sorting unit has $N/4 \times (\log_2(N)) \times (\log_2(N) - 1) + N - 1$ CAE blocks.

2.3 Designing Large Sorting Networks

Based on the idea behind the bitonic and odd-even merge algorithms, large sorting units can be built using large merging units [8] that consist of multiple

[1]BM-2 and OEM-2 units have the same structure.

CAE stages of increasingly larger size. Table 2.1 summarizes the required number of CAE blocks and stages for bitonic and odd-even merge sorting units. Both N-input bitonic and odd-even merge sorting units have a time complexity (depth) of $O(\log_2^2(N))$ CAE stages and have an area complexity of $O(N \times \log_2^2(N))$ CAE blocks. An N-input, N-output bitonic or odd-even merge complete sorting unit is composed of $(\log_2 N) \times (\log_2 N + 1)/2$ CAE stages, where $N = 2^n$. However, the required number of CAE blocks differs for each type of sorting unit. Bitonic and odd-even merge sorting unit with 2^n inputs and 2^n outputs have $2^{n-2} \times n \times (n+1)$ and $2^{n-2} \times n \times (n-1) + 2^n - 1$ CAE blocks, respectively. Thus, odd-even merge sorting units have lower resource requirements than bitonic sorting units, but may have more complex wiring. The difference in the number of CAE blocks between bitonic and odd-even merge sorting units is $2^{n-1} \times (n-2) + 1$, which shows that the difference in the number of CAE blocks increases linearly with the number of inputs.

3 PROPOSED PARTIAL SORTING AND MAX-SET-SELECTION UNITS

In many applications, it is not necessary to return all of the sorted inputs. Applications often only need to determine the M largest or M smallest numbers from N inputs, where $M < N$ and M and N are both integer powers of two ($M = 2^m$, $N = 2^n$). Partial sorters provide the 2^m largest values in sorted order, and max-set-selection units provide the 2^m largest values in arbitrary order. Partial sorters and max-set-selection units are key components in many applications such as HEP and multimedia applications. For example, in the Large Hadron Collider [35], built by the European Organization for Nuclear Research (CERN), low-latency max-set-selection units identify important particle interactions that correspond to high-energy collisions [18, 55]. In multimedia applications, partial sorting units speed up data sorting algorithms [17]. Moreover, auxiliary max-set-selection units can cooperate with general-purpose processing units in embedded and database management systems to accelerate data search and sort algorithms. In cases such as this, Batcher's algorithms can be optimized to generate only the 2^m largest numbers from 2^n inputs with less latency and fewer CAE blocks than a complete sorting network.

In this chapter, we propose a modular technique based on Batcher's algorithms to design units that return only the 2^m largest values from 2^n inputs. In Chapter 3.1, we focus on the case that only the four largest values are produced by the max-set-selection unit without regard to output order. We also extend our technique to partial sorting units to return outputs in ascending order. In Chapter 3.2, we generalize our method so the proposed algorithm is applicable for any number of outputs that is an integer power of two and where results can be returned either in arbitrary or ascending order.

3.1 4-Output Max-set-selection and Partial Sorting Units

We first discuss 8-to-4 max-set-selection units and then extend our technique to larger 2^n-to-4 max-set-selection and partial sorting units. To design 2^n-to-4 max-set-selection units, we take advantage of the fact that only the four largest inputs are needed, in no particular order, to decrease the resource requirements and the number of CAE stages.

3.1.1 8-to-4 Max-set-selection Units

To illustrate our technique, we first present the design of a refined 8-input bitonic sorting unit, called an 8-to-4 bitonic max-set-selection unit, that only returns the four largest numbers. A special feature of bitonic sequences is that performing Max operations on two sorted sequences (one increasing and the other one decreasing) each of K numbers, generates two new sequences of K numbers in which numbers in one sequence are all less than numbers in the other sequence. In BM-K units, the first level of parallel CAE blocks partitions the input numbers into two (K/2)-number sub-sets: the smaller numbers and the larger numbers. However, the first level of parallel CAE blocks in OEM units must be rewired to correctly generate the smaller and larger subsets of numbers. Figs. 3.1 and 3.2 show 8-to-4 max-set-selection units that use bitonic and odd-even merge algorithms, respectively. As shown in Fig. 3.1, the BM-8 unit in Fig. 2.3 is replaced with a level of Max units that has the same wiring as the first level of parallel CAE blocks in the BM-8 unit. Fig. 3.2 illustrates that the OEM-8 unit in Fig. 2.5 is replaced with a level of Max units with wirings that differ from the first level of parallel CAE blocks in the OEM-8 unit. These modifications decrease the required number of CAE stages from six in 8-input sorting units to four in 8-to-4 max-set-selection units.

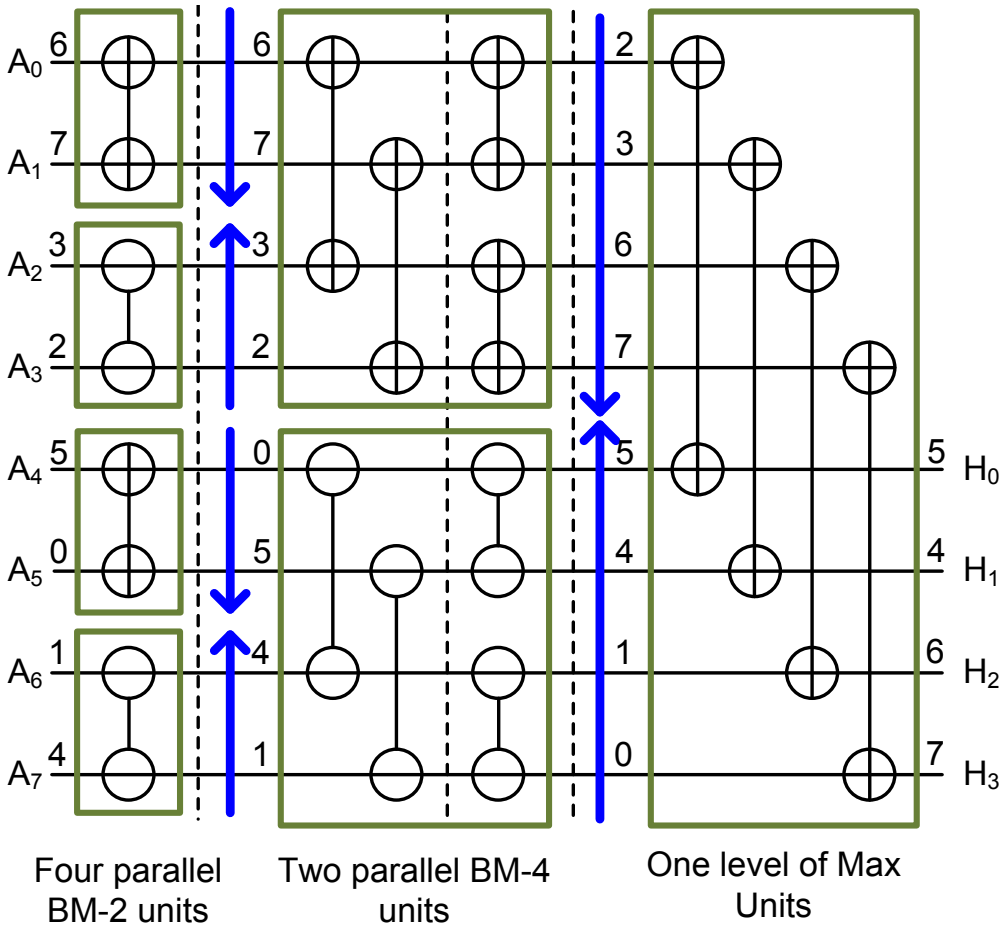

Figure 3.1: The CAE network for an 8-to-4 bitonic max-set-selection unit with four CAE stages and 16 CAE blocks.

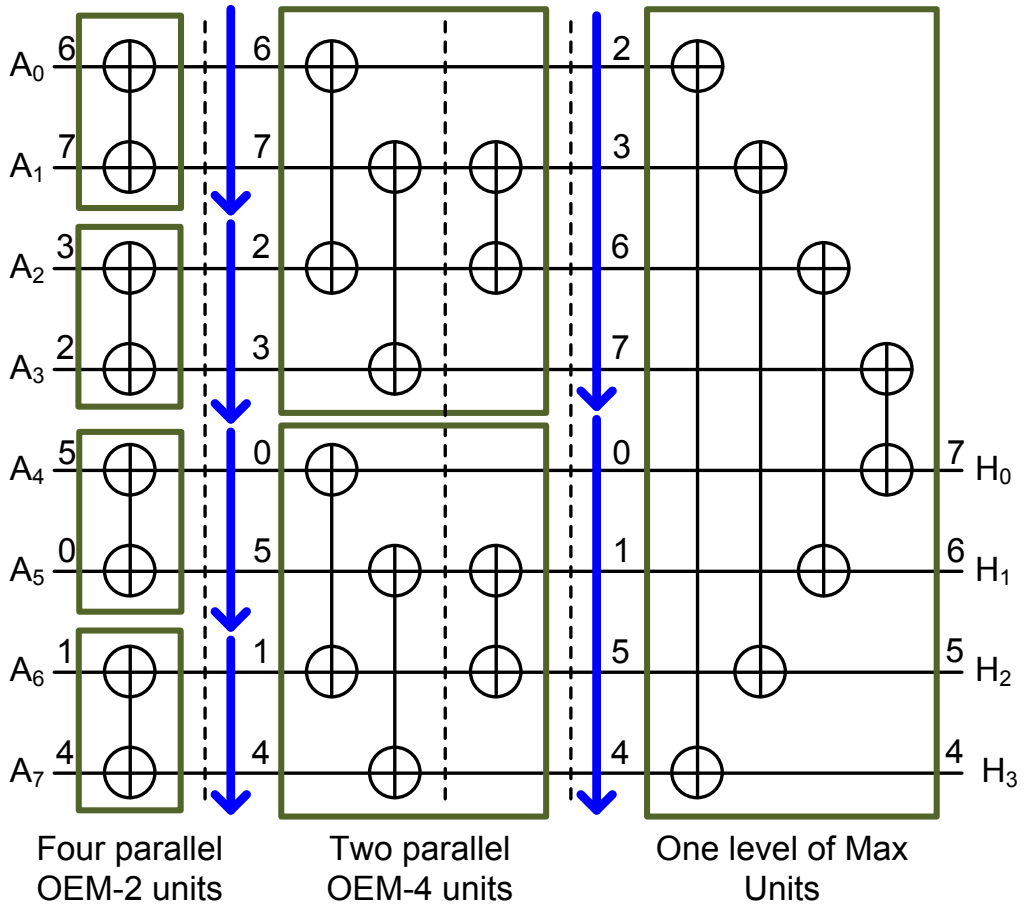

Figure 3.2: The CAE network for an 8-to-4 odd-even merge max-set-selection unit with four CAE stages and 14 CAE blocks.

3.1.2 BM-8-to-4 and 8-to-4 Partial Sorting Units

The 8-to-4 max-set-selection units, however, cannot be used directly to form larger sorting or max-set-selection units, because the outputs of the 8-to-4 max-set-selection units in Fig. 3.1 and Fig. 3.2 are not in a specific order. Since inputs to BM units should be a bitonic sequence and inputs to OEM units should be two ascending sequences, these designs cannot be connected directly to other BM or OEM units when desiging larger sorters.

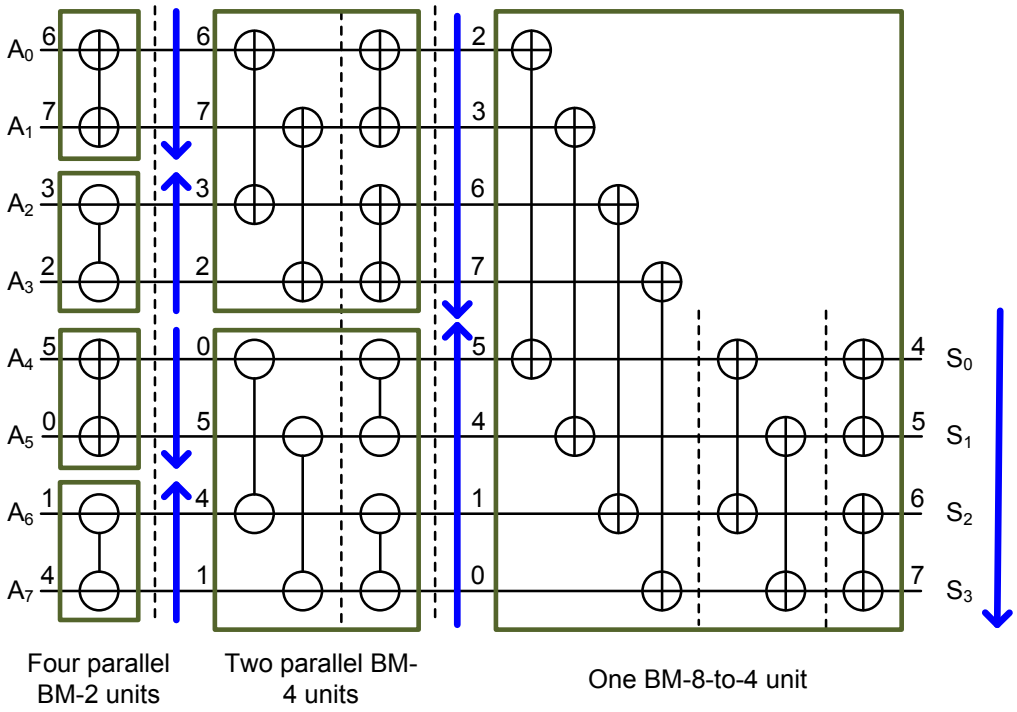

Figure 3.3: The CAE network for an 8-to-4 bitonic partial sorting unit with six CAE stages and 20 CAE blocks.

To solve this problem, we have designed a new merging unit called a BM-8-to-4 unit, shown as the right-most block in Figs. 3.3 and 3.4. A BM-8-to-4 unit is an 8-input bitonic merging unit that outputs only the four largest values in either ascending $\oplus$ or descending $\ominus$ order. Fig. 3.3 depicts our proposed 8-to-4 bitonic partial sorting unit that returns the four largest values from its eight inputs in ascending order. Compared to the 8-input bitonic sorting unit shown in Fig. 2.3, it needs fewer CAE blocks and the BM-8 unit is replaced with a BM-8-to-4 unit. A descending 8-to-4 partial sorting unit has a similar structure. A BM-8-to-4 unit has a level of four parallel CAE blocks followed by a BM-4 unit. With this approach, the output of an 8-to-4 bitonic partial sorting unit can be fed to other bitonic merging units.

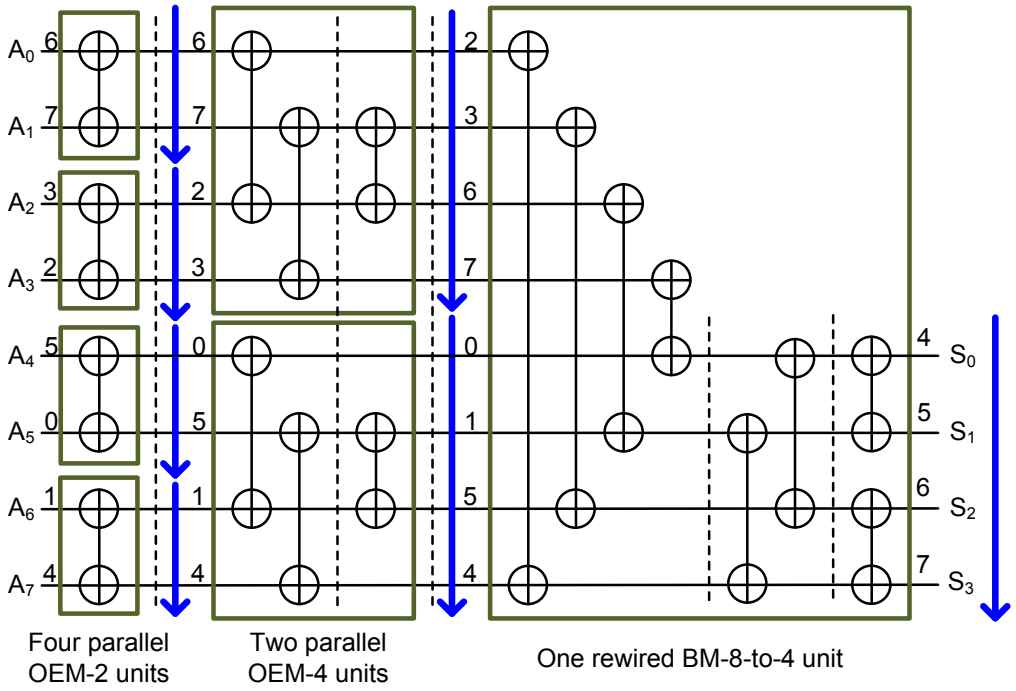

Figure 3.4: The CAE network for an 8-to-4 odd-even merge partial sorting unit with six CAE stages and 18 CAE blocks.

Fig. 3.4 depicts our proposed 8-to-4 odd-even merge partial sorting unit that returns the four largest values from its eight inputs in ascending order. Compared to the 8-input odd-even merge sorting unit in Fig. 2.5, it needs fewer CAE blocks and the OEM-8 unit is replaced with a BM-8-to-4 unit. In fact, the partial sorting unit shown in Fig. 3.4 is a hybrid unit composed of bitonic and odd-even merging units. Since a bitonic merging unit takes only a bitonic sequence, the inputs to the BM-8-to-4 unit in Fig. 3.4 are rewired to convert the two sorted sequences to a bitonic sequence. This way, the output of odd-even merging units can be fed to bitonic merging units. We could also propose an OEM-8-to-4 unit to avoid the rewiring technique. However, since an OEM-8-to-4 unit requires more registers than a BM-8-to-4, only BM-2^{k+1}-to-2^k units are used throughout the thesis. Note

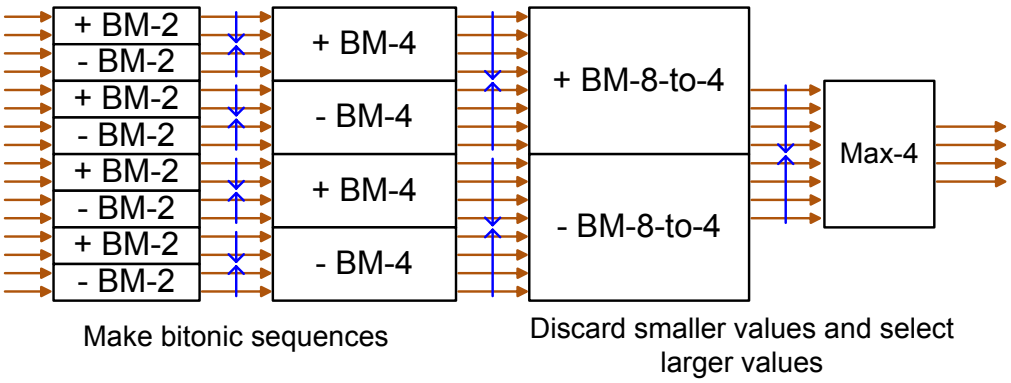

Figure 3.5: A 16-to-4 bitonic max-set-selection unit.

that only increasing BM-8-to-4 units are used in our proposed odd-even merge sorting unit, while both increasing and decreasing BM-8-to-4 units are used in the bitonic sorting unit.

3.1.3 2^n-to-4 Max-set-selection and Partial Sorting Units

To design larger 2^n-to-4 max-set-selection units, we can take advantage of the fact that BM-8-to-4 units can be combined to make larger units. Since BM-8-to-4 units generate sorted outputs, these outputs can feed other BM-8-to-4 units. We can build larger 2^n-to-4 bitonic max-set-selection units using BM-2, BM-4, BM-8-to-4, and Max-4 units. We can also build larger 2^n-to-4 odd-even merge max-set-selection units using OEM-2, OEM-4, BM-8-to-4, and Max-4 units. BM-2, BM-4, BM-8, BM-8-to-4, and Max-4 units require 1, 4, 12, 8, and 4 CAE blocks, respectively. OEM-4 and OEM-8 units require 3 and 9 CAE blocks, respectively.

Figs. 3.5 shows the structures of 16-to-4 bitonic max-set-selection units that utilize multiple BM-8-to-4 units. Fig. 3.6 shows the structure of a 32-to-4 bitonic max-set-selection unit that has 10 CAE stages. Fig. 3.7 shows the structure of a 32-to-32 bitonic sorting unit that has 15 CAE

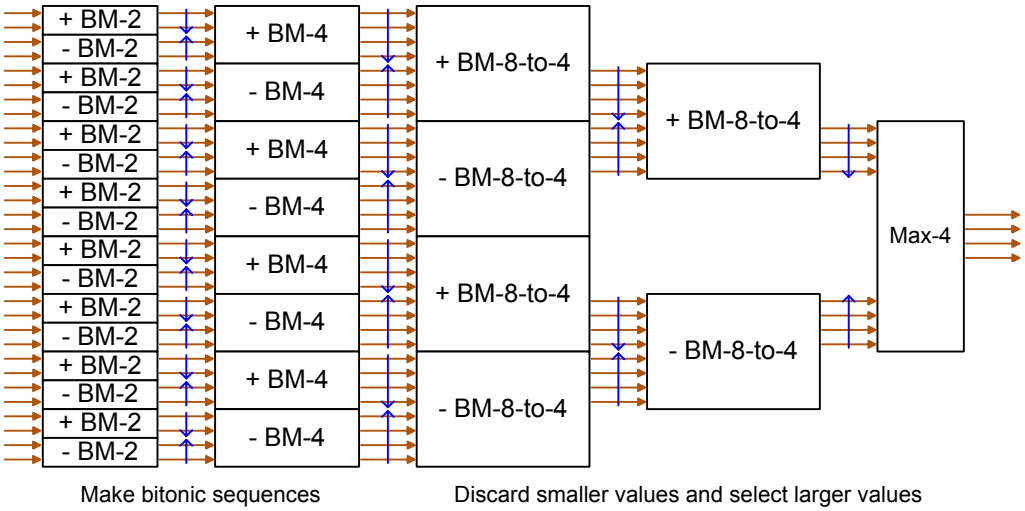

Figure 3.6: A 32-to-4 bitonic max-set-selection unit.

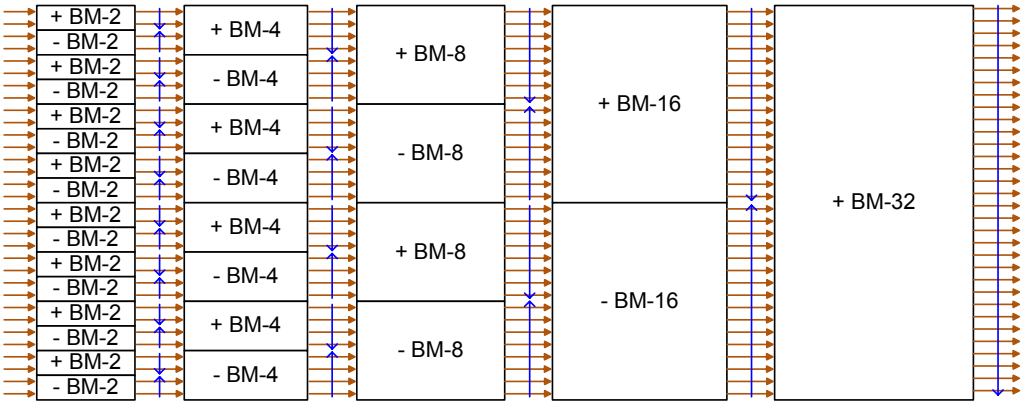

Figure 3.7: A 32-to-32 bitonic sorting unit.

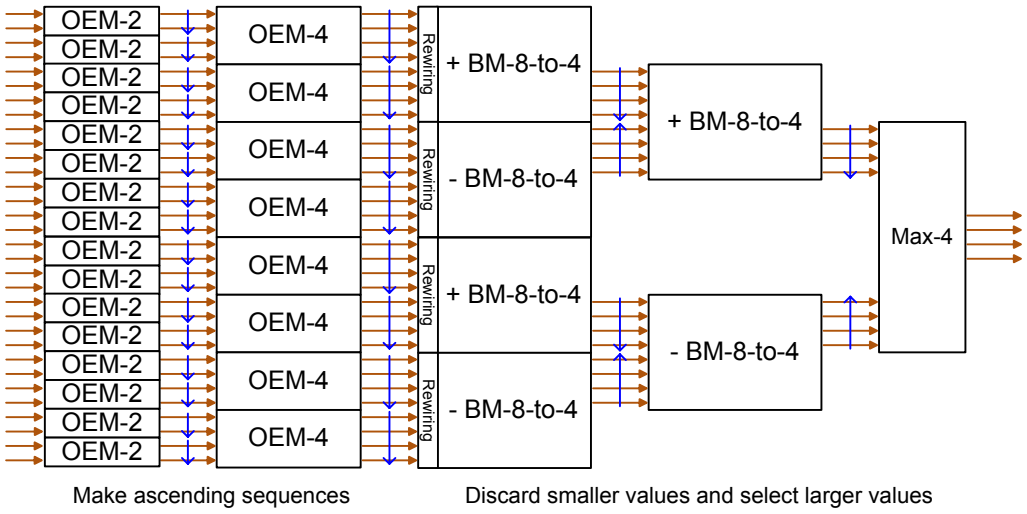

Figure 3.8: A 32-to-4 odd-even merge max-set-selection unit.

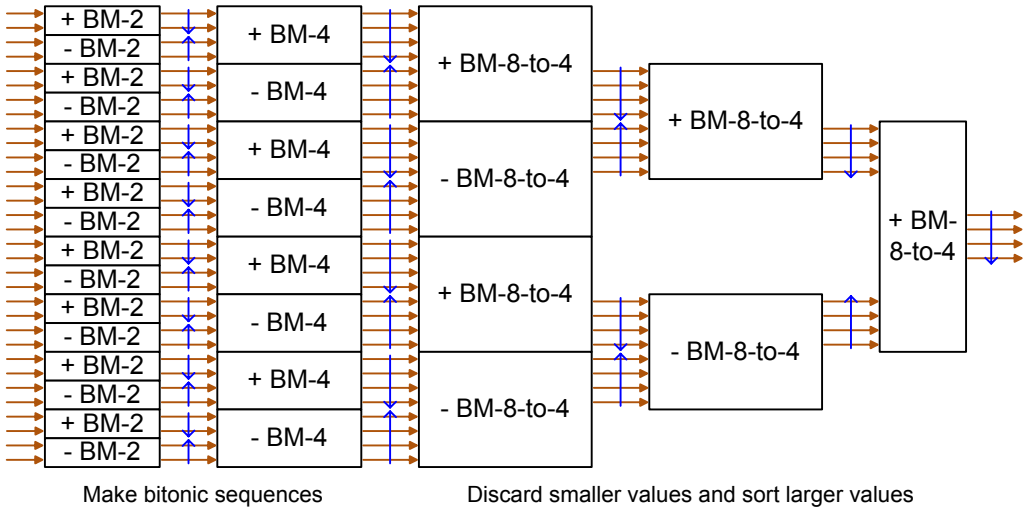

Figure 3.9: A 32-to-4 bitonic partial sorting unit.

stages. In these designs, smaller numbers are discarded in stages as early as possible to reduce the total number of CAE stages and CAE blocks. A 16-to-4 bitonic max-set-selection unit has one level of parallel BM-8-to-4 units, while a 32-to-4 bitonic max-set-selection unit employs two levels of parallel BM-8-to-4 units.

Fig. 3.8 depicts the high-level structure of a 32-to-4 odd-even merge max-set-selection unit that utilizes several OEM and BM-8-to-4 units. Unlike bitonic max-set-selection and partial sorting units, odd-even merge units only use increasing merging units, which is an advantage in terms of simplicity of design. On the other hand, as shown in Fig. 3.8, the outputs of OEM-4 units are rewired to feed BM-8-to-4 units, which is a disadvantage. In general, 2^n-to-4 max-set-selection units have $n-3$ levels of parallel BM-8-to-4 units for $n > 3$.

To make a 2^n-to-4 partial sorting unit, the last level of a 2^n-to-4 max-set-selection unit, which is a Max-4 unit, is replaced by a BM-8-to-4 unit to generate outputs in sorted order. This increases the number of CAE stages and the number of CAE blocks by 2 and 4, respectively. Fig. 3.9 shows the structure of a 32-to-4 bitonic partial unit (compare this with Fig. 3.6).

Table 3.1 compares the resource requirements of sub-units used in 2^n-to-4 max-set-selection and 2^n-to-2^n sorting units. Table 3.2 shows the resource requirements and the number of CAE stages for our proposed 2^n-to-4 and 2^n-to-8 max-set-selection units. The required number of CAE stages for a 2^n-to-4 bitonic max-set-selection unit can be calculated based on the fact that it is composed of a level of parallel BM-2 units, a level of parallel BM-4 units, $n-3$ levels of parallel BM-8-to-4 units, and a Max-4 unit. Similarly, the required number of CAE blocks can be calculated based on the fact that there are 2^{n-1} BM-2 units, 2^{n-2} BM-4 units, $2^{n-2}-2$ BM-8-to-4 units, and one Max-4 unit in a 2^n-to-4 bitonic max-set-selection unit. Both 2^n-to-4 bitonic and odd-even merge max-set-selection units have $3 \times n - 5$ CAE stages for $n \geqslant 3$. 2^n-to-4 bitonic and odd-even merge max-set-selection units

require $7 \times 2^{n-1} - 12$ and $13 \times 2^{n-2} - 12$ CAE blocks, respectively. Thus, 2^n-to-4 max-set-selection and partial sorting units require many fewer CAE stages and CAE blocks than conventional 2^n-input complete sorting units for large values of n. For example, while a 256-input sorting unit has 36 CAE stages, a 256-to-4 max-set-selection unit has 19 CAE stages, and a 256-to-4 partial sorting unit has 21 CAE stages. As shown in Table 2.1, a 256-input bitonic sorting units requires 4,608 CAE blocks. However, a 256-to-4 bitonic max-set-selection unit and a 256-to-4 bitonic partial sorting unit require 884 and 888 CAE blocks, respectively. Similarly, as shown in Table 2.1, a 256-input odd-even merge sorting unit requires 3,839 CAE blocks. However, a 256-to-4 odd-even merge max-set-selection unit and a 256-to-4 odd-even merge partial sorting unit require 820 and 824 CAE blocks, respectively. This indicates significant improvements of max-set-selection compared to the conventional sorting units in terms of both the number of CAE stages and the required number of CAE blocks.

Table 3.2 shows how max-set-selection units can be built from BM/OEM and Max units. For instance, to implement a 256-to-4 max-set-selection unit, 128 BM-2/OEM-2 units, 64 BM-4/OEM-4 units, 62 BM-8-to-4 units, and a Max-4 unit are needed. In general, odd-even merge max-set-selection units need fewer CAE blocks than the corresponding bitonic max-set-selection units, but bitonic max-set-selection units have more regular structures and are easier to design. Table 3.2 indicates that the fastest 256-to-4 max-set-selection units have a latency of 19 clock cycles (assuming each CAE stage takes one clock cycle). In comparison, conventional 256-input sorting units require 36 clock cycles to sort all 256 data values.

We also investigated constructing 256-to-4 max-set-selection units using 8-to-4, 16-to-4, 32-to-4, 64-to-4, and 128-to-4 max-set-selection units, as shown in Table 3.3. Tables 3.2 and 3.3 indicate that the fastest 256-to-4 max-set-selection units have a latency of 19 clock cycles (assuming each stage takes one clock cycle). In comparison, conventional 256-input sorting

Table 3.1: Sub-units used in 2^n-to-4 max-set-selection and 2^n-to-2^n sorting units (10-bit unsigned data width)

Sub-unit	CAE Blocks	CAE Stages	Slice LUTs	Slice Regs
BM-2	1	1	30	20
Max-4	4	1	60	40
BM-4	4	2	100	80
BM-8	12	3	300	240
BM-8-to-4	8	3	160	120
BM-16	32	4	800	640
BM-32	80	5	2,000	1,600
BM-64	192	6	4,800	3,840
BM-128	448	7	11,200	8,960
BM-256	1,024	8	25,600	20,480

units require 36 clock cycles to sort all 256 data values. In addition, the odd-even merge max-set-selection unit design shown in the last row of Table 3.2 requires fewer CAE blocks than the designs introduced in Table 3.3. In general, odd-even merge max-set-selection units need fewer CAE blocks than bitonic max-set-selection units, but bitonic max-set-selection units have more regular structures and are easier to design.

Table 3.2: Structure and number of CAE stages and CAE blocks for 2^n-to-4 and 2^n-to-8 bitonic and odd-even merge max-set-selection units made up of smaller merging units. The numbers in parentheses under "Structure" show the required number of each unit.

Max-set-selection	Structure		# of CAE Stages	# of CAE Blocks	
	Bitonic	Odd-even		Bitonic	Odd-even
8-to-4	BM-2(4)→ BM-4(2)→ Max-4(1)	OEM-2(4)→ OEM-4(2)→ Max-4(1)	4	16	14
16-to-4	BM-2(8)→ BM-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(8)→ OEM-4(4)→ BM-8-to-4(2)→ Max-4(1)	7	44	40
16-to-8	BM-2(8)→ BM-4(4)→ BM-8(2)→ Max-8(1)	OEM-2(8)→ OEM-4(4)→ OEM-8(2)→ Max-8(1)	7	56	46
32-to-4	BM-2(16)→ BM-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(16)→ OEM-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	10	100	92
32-to-8	BM-2(16)→ BM-4(8)→ BM-8(4)→ BM-16-to-8(2)→ Max-8(1)	OEM-2(16)→ OEM-4(8)→ OEM-8(4)→ BM-16-to-8(2)→ Max-8(1)	11	144	124
64-to-4	BM-2(32)→ BM-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(32)→ OEM-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	13	212	196
64-to-8	BM-2(32)→ BM-4(16)→ BM-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	OEM-2(32)→ OEM-4(16)→ OEM-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	15	320	280
128-to-4	BM-2(64)→ BM-4(32)→ BM-8-to-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(64)→ OEM-4(32)→ BM-8-to-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	16	436	404
128-to-8	BM-2(64)→ BM-4(32)→ BM-8(16)→ BM-16-to-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	OEM-2(64)→ OEM-4(32)→ OEM-8(16)→ BM-16-to-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	19	672	592
256-to-4	BM-2(128)→ BM-4(64)→ BM-8-to-4(32)→ BM-8-to-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	OEM-2(128)→ OEM-4(64)→ BM-8-to-4(32)→ BM-8-to-4(16)→ BM-8-to-4(8)→ BM-8-to-4(4)→ BM-8-to-4(2)→ Max-4(1)	19	884	820
256-to-8	BM-2(128)→ BM-4(64)→ BM-8(32)→ BM-16-to-8(16)→ BM-16-to-8(8)→ BM-16-to-8(4)→ BM-16-to-8(2)→ Max-8(1)	OEM-2(128)→ OEM-4(64)→ OEM-8(32)→ OEM-16-to-8(16)→ OEM-16-to-8(8)→ OEM-16-to-8(4)→ OEM-16-to-8(2)→ Max-8(1)	23	1372	1216

Table 3.3: The structure, the number of CAE stages, and resource requirements for 256-to-4 max-set-selection units made up of smaller max-set-selection units. The numbers in parentheses show the required number of each max-set-selection unit.

256-to-4 Max-set-selection	Structure	# of CAE Stages	# of CAE Blocks	
			Bitonic	Odd-even
Using 8-to-4	8-to-4(32)→ 8-to-4(16)→ 8-to-4(8)→ 8-to-4(4)→ 8-to-4(2)→ 8-to-4(1)	$6 \times 4 = 24$	$63 \times 16 = 1008$	$63 \times 14 = 882$
Using 16-to-4	16-to-4(16)→ 16-to-4(4)→ 16-to-4(1)	$3 \times 7 = 21$	$21 \times 44 = 924$	$21 \times 40 = 840$
Using 32-to-4	32-to-4(8)→ 32-to-4(1)	$2 \times 10 = 20$	$9 \times 100 = 900$	$9 \times 92 = 828$
Using 64-to-4	64-to-4(4)→ 16-to-4(1)	$13 + 7 = 20$	$4 \times 212 + 44 = 936$	$4 \times 196 + 40 = 824$
Using 128-to-4	128-to-4(2)→ 8-to-4(1)	$16 + 4 = 20$	$2 \times 436 + 16 = 888$	$2 \times 404 + 14 = 822$

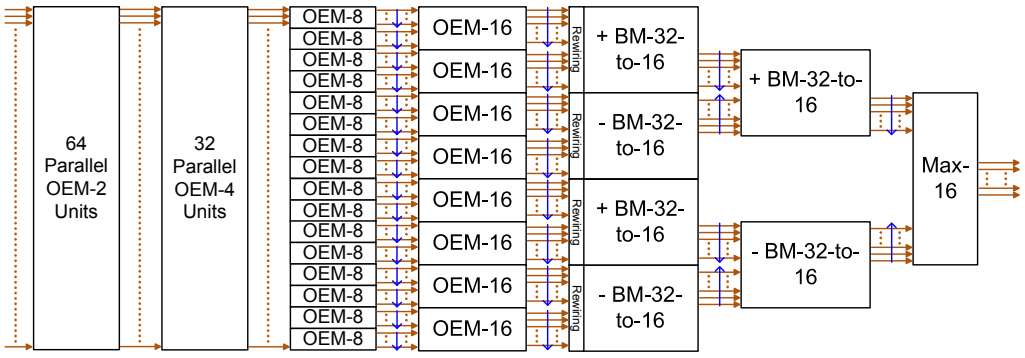

Figure 3.10: A 128-to-16 odd-even merge max-set-selection unit.

3.2 2^n-to-2^m Max-set-selection and Partial Sorting Units

Our proposed techniques can be extended to cover a wide range of sorting and max-set-selection units. In the previous section, 2^n-to-4 max-set-selection units (with $n > 3$) utilize BM-8-to-4 units to select the four largest values out of a bitonic sequence with eight values. The sorted outputs of BM-8-to-4 units are then combined to form bitonic sequences that are fed to the next stage of BM-8-to-4 units or a Max-4 unit. The smallest values are discarded in the earlier stages to reduce resources and latency.

3.2.1 Modular Max-set-seletion Units

A similar approach can be used to design 2^n-to-2^m max-set-selection units with BM-2^{m+1}-to-2^m merging units. Starting from unsorted inputs and by following Batcher's algorithms, small sorted sequences are constructed using either BM or OEM units. When 2^{n-m} sorted sequences of length 2^m are generated, each pair of sorted sequences of length 2^m is fed to a BM-2^{m+1}-to-2^m unit to produce the 2^m largest values in sorted order and discard the 2^m smallest values. Using BM-2^{m+1}-to-2^m units, this process continues

until only two sorted sequences of length 2^m are left. These two sorted sequences contain the 2^{m+1} largest values from the 2^n input values. In the final stage, a Max-2^m unit returns the 2^m largest values in arbitrary order. With this approach, a 2^n-to-2^m bitonic or odd-even merge max-set-selection unit with $n > m$ has a total of $(2^{n-m} - 2)$ BM-2^{m+1}-to-2^m merging units in $n - m - 1$ levels, where each BM-2^{m+1}-to-2^m unit is composed of a level of 2^m parallel CAE blocks and a BM-2^m unit[1]. For example, a bitonic 128-to-16 max-set-selection unit has 64 BM-2, 32 BM-4, 16 BM-8, eight BM-16, six BM-32-to-16, and one Max-16 units, where each BM-32-to-16 unit has a level of 16 parallel CAE blocks and a BM-16 unit. Similarly, a 128-to-16 odd-even merge max-set-selection unit, shown in Fig. 3.10, has 64 OEM-2, 32 OEM-4, 16 OEM-8, eight OEM-16, six BM-32-to-16, and one Max-16 unit. Table 3.2 shows the structure of 2^n-to-4 and 2^n-to-8 max-set-selection units.

3.2.2 Modular Partial Sorting Units

We can easily modify max-set-selection units to produce their outputs in ascending order rather than arbitrary order. To design a 2^n-to-2^m partial sorting unit that takes 2^n inputs and returns 2^m sorted outputs, the Max-2^m unit in a 2^n-to-2^m max-set-selection unit is replaced with a BM-2^{m+1}-to-2^m unit, producing outputs in ascending order. Thus, a 2^n-to-2^m partial sorting unit has a total of $(2^{n-m} - 1)$ BM-2^{m+1}-to-2^m merging units. This increases the number of CAE stages and the number of CAE blocks for a 2^n-to-2^m sorting unit over its corresponding max-set-selection unit by m and $2^{m-1} \times m$, respectively.

[1] A BM-2-to-1 is basically a Max-1 unit

3.3 Other Extensions

3.3.1 Other Input Quantities

In situations where the number of inputs, N, is not an integer power of two, two approaches can be taken to design customized merging and max-set-selection units. The easier approach is to use a max-set-selection unit with 2^n inputs, where 2^n is the smallest integer power of two larger than N. The unused inputs can be set to zero so that the synthesis tool eliminates unnecessary logic. The other approach, which does not sacrifice area for simplicity, is to use design techniques proposed in [4, 25, 32, 37, 40]. Although these techniques are not straight-forward, they can be applied to the design of max-set-selection units. Since these techniques are based on an algorithmic approach, they are able to further reduce logic, leading to a more area-efficient designs.

3.3.2 Other Output Quantities

In situations where the number of outputs, M, is not an integer power of two, a straight-forward approach is to design merging units with 2^m outputs, where 2^m is the smallest integer power of two larger than M. The intermediate stages for this approach are the same as those for the regular 2^m-output max-set-selection units. However, the the difference is that only the M largest outputs are produced in the final stage. The other approach, which has fewer resource requirements, is to design merging units with M outputs. In this approach, BM-2M-to-M units should be designed, which for each different value of M, designers need to devise a new CAE structure for the given merging unit. Using either approaches, the latency of a 2^n-to-M max-set-selection unit is the same as that of a 2^n-to-2^m max-set-selection unit, where $(2 \times M) > 2^m \geqslant M$.

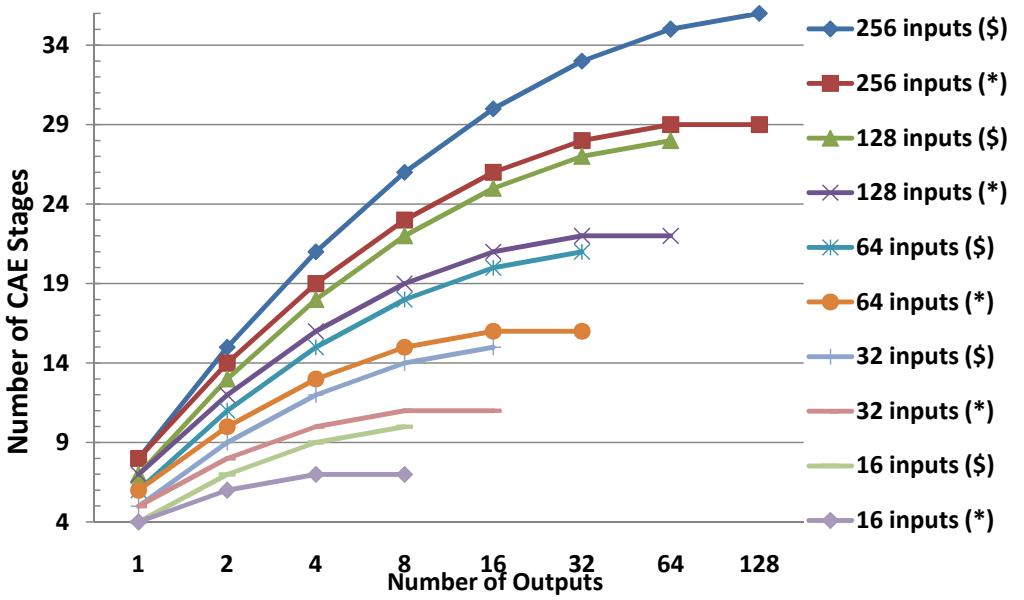

Figure 3.11: The number of CAE stages for 2^n-to-2^m partial sorting ($) and max-set-selection (*) units.

3.4 Analysis

We can analyze the number of CAE stages and CAE blocks for 2^n-to-2^m max-set-selection and sorting units using our proposed approach. Fig. 3.11 shows the number of CAE stages for bitonic and odd-even merge max-set-selection and partial sorting units. In this figure, * indicates the required number of CAE stages when outputs have arbitrary order (max-set-selection units) and $ indicates the number of stages when outputs are sorted (sorting units). In all cases, bitonic and odd-even merge units have the same number of CAE stages. The number of CAE stages in max-set-selection units and partial sorting units is $n(m+1) - m(m+3)/2$ and $(2n-m)(m+1)/2$, respectively. Thus, N-to-M max-set-selection and partial sorting units have a time complexity of $O(\log_2 N \times \log_2 M)$, where $N = 2^n$ and $M = 2^m$. For a 2^m-output max-set-selection or partial sorting unit, doubling the number of inputs increases the number of CAE stages by $m+1$. For a 2^n-input

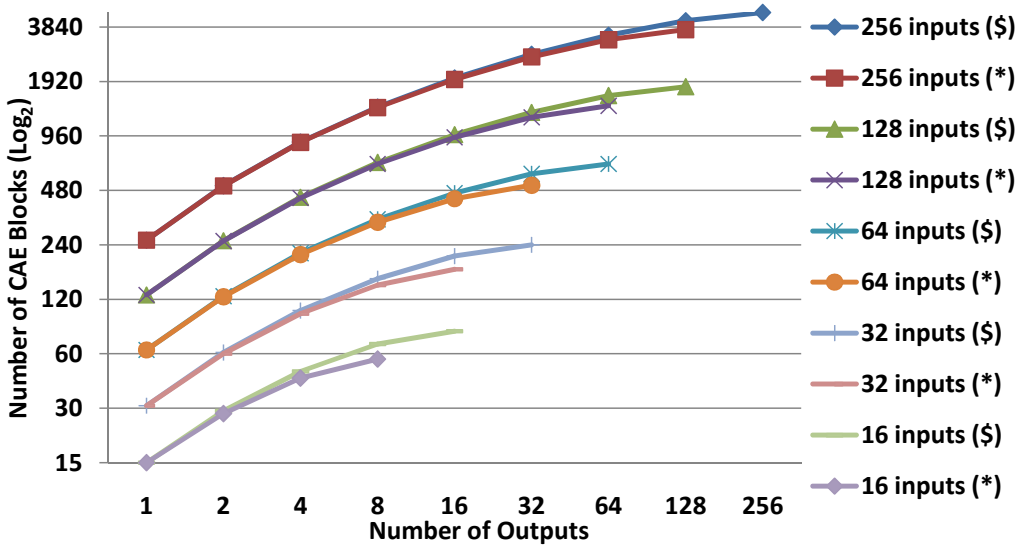

Figure 3.12: The number of CAE blocks for 2^n-to-2^m bitonic partial sorting ($) and max-set-selection (*) units.

max-set-selection unit and a 2^n-input partial sorting unit, doubling the number of outputs from 2^m to 2^{m+1} increases the number of CAE stages by $n-m-2$ and $n-m-1$, respectively. The difference in the number of CAE stages between sorting units and max-set-selection units increases logarithmically with the number of outputs.

Fig. 3.12 shows the required number of CAE blocks for 2^n-to-2^m bitonic partial sorting and max-set-selection units as n and m are varied. The number of CAE blocks in bitonic max-set-selection and partial sorting units is $(m(m+3)+4)\times 2^{n-2}-2^m\times(m+1)$ and $(m(m+3)+4)\times 2^{n-2}-2^{m-1}\times(m+2)$, respectively. Keeping the number of outputs fixed, the number of CAE blocks increases linearly with the number of inputs. Keeping the number of inputs fixed, the number of CAE blocks increases sub-linearly with the number of outputs. The difference in the number of CAE blocks between partial sorting units and max-set-selection units increases logarithmically with the number of outputs.

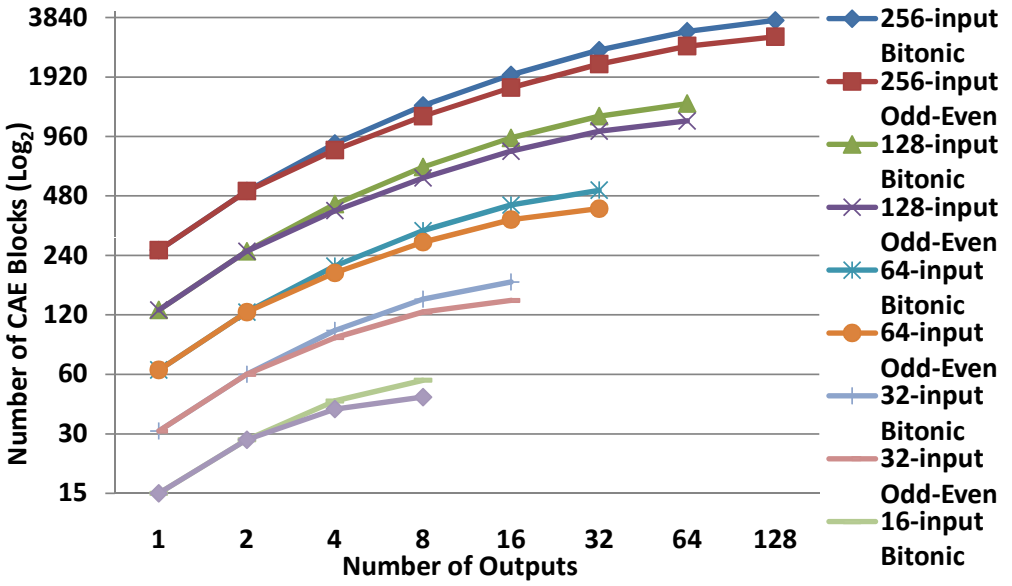

Figure 3.13: The number of CAE blocks for 2^n-to-2^m bitonic and odd-even merge max-set-selection units.

Fig. 3.13 compares the required number of CAE blocks for 2^n-to-2^m bitonic and odd-even merge max-set-selection units as n and m are varied. The number of CAE blocks in odd-even merge max-set-selection and partial sorting units is $(m(m+3)+4)\times 2^{n-2}-m\times 2^{n-1}+(1-2^{-m})\times 2^n-2^m\times(m+1)$ and $(m(m+3)+4)\times 2^{n-2}-m\times 2^{n-1}+(1-2^{-m})\times 2^n-2^{m-1}\times(m+2)$, respectively. Thus, N-to-M bitonic and odd-even merge max-set-selection and partial sorting units have an area complexity of $O(N\times \log_2^2(M))$. Fig. 3.14 compares the number of CAE blocks for 2^n-to-2^m bitonic and odd-even merge sorting units. In both Figs. 3.13 and 3.14, For all the units, the number of CAE blocks increases linearly with the number of inputs. When the number of outputs is one or two, bitonic and odd-even merge max-set-selection and partial sorting units require the same number of CAE blocks. However, as the number of inputs increases, the benefit of using the odd-even merge algorithm over the bitonic algorithm increases in

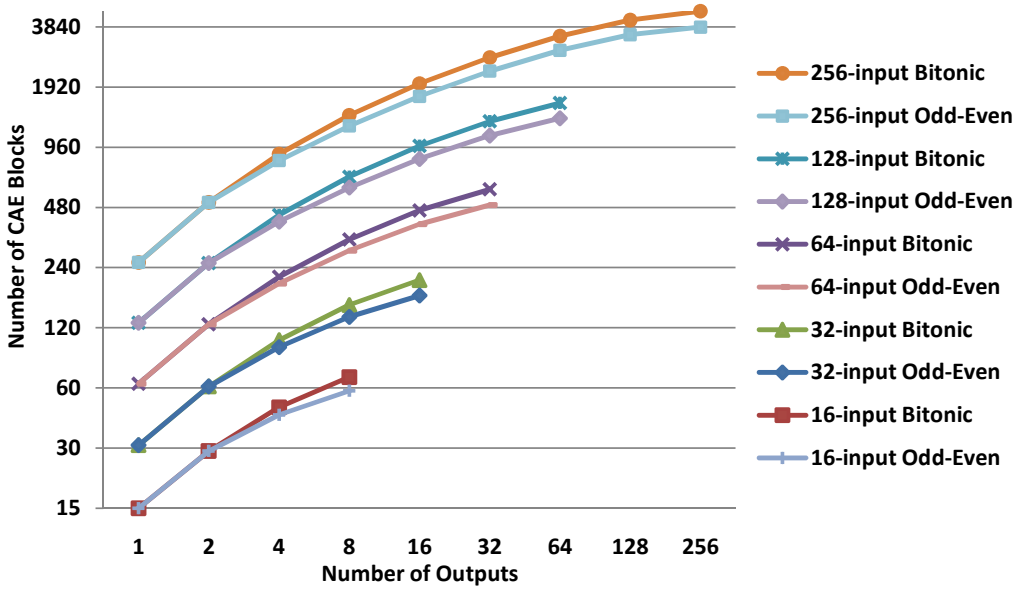

Figure 3.14: The number of CAE blocks for 2^n-to-2^m bitonic and odd-even merge partial sorting units.

terms of the required number of CAE blocks for both partial sorting and max-set-selection units.

4 RESULTS

To assist with analyzing implementations of our proposed techniques, we developed Verilog register transfer level (RTL) models for 2^n-to-2^m partial sorting and max-set-selection units. The Verilog models are fully parameterized to provide the flexibility needed to design and analyze a wide range of sorting and max-set-selection units. The designer can change (add or remove) each level of pipeline registers to get a design with a different latency, resource requirements, and frequency. This feature helps achieve the desired throughout and latency. The models are composed of small, verified building blocks to simplify the design process and facilitate testing.

4.1 ASIC Implementation

The proposed designs are synthesized using the Synopsys design compiler vB 2008.09 SP3 and a TSMC 65-nm standard-cell library. The designs are pipelined and all outputs are registered. For all the synthesis results, the parameterizable data width (i.e, the CAE width), which can be easily changed, is set to ten unsigned bits, which is commonly used in HEP applications. Tables 4.1 and 4.2 show post-place-and-route results for 2^n-to-4 and 2^n-to-8 max-set-selection units, respectively. We report implementation results for three different pipeline structures (depths) for each bitonic and odd-even max-set-selection unit: one CAE stage between pipeline registers, two CAE stages between pipeline registers, and three CAE stages between pipeline registers. The last column of Tables 4.1 and 4.2 lists the end-to-end latency for each design. All max-set-selection units can achieve high frequencies due to the regular pipelined structure of the designs.

Table 4.1: Performance and resource requirements of 2^n-to-4 bitonic and odd-even merge max-set-selection units with 10-bit unsigned CAE blocks using a TSMC 65-nm standard-cell library

Max-set-selection	Pipeline Depth	Frequency (MHz)		Combinational Area (μm^2)		Non-comb. Area (μm^2)		End-to-end Latency (ns)	
		Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even
16-to-4	7	2,941	2,941	5,400	4,862	8,781	8,317	2.38	2.38
16-to-4	4	1,851	2,000	7,881	7,171	4,282	4,671	2.16	2.00
16-to-4	3	1,449	1,449	8,240	6,852	3,514	3,540	2.07	2.07
32-to-4	10	2,857	2,857	13,696	11,229	20,008	17,962	3.50	3.50
32-to-4	5	1,886	1,886	16,681	16,174	11,503	10,648	2.65	2.65
32-to-4	4	1,428	1,449	18,960	18,695	7,764	8,241	2.80	2.76
64-to-4	13	2,777	2,857	24,888	20,140	39,633	37,324	4.68	4.55
64-to-4	7	1,785	1,851	40,880	30,914	22,922	22,204	3.92	3.78
64-to-4	5	1,315	1,449	42,507	35,227	17,143	18,172	3.80	3.45
128-to-4	16	2,702	2,702	46,029	48,178	70,695	68,465	5.92	5.92
128-to-4	8	1,785	1,754	83,975	70,339	37,773	45,253	4.48	4.56
128-to-4	6	1,369	1,315	89,493	69,588	37,164	35,575	4.38	4.56
256-to-4	19	2,702	2,702	94,606	85,178	142,308	137,457	7.03	7.03
256-to-4	10	1,754	1,754	166,578	144,062	90,221	87,715	5.70	5.70
256-to-4	7	1,298	1,315	150,596	139,322	64,517	53,936	5.39	5.32

Table 4.2: Performance and resource requirements of 2^n-to-8 bitonic and odd-even merge max-set-selection units with 10-bit unsigned CAE blocks using a TSMC 65-nm standard-cell library

Max-set-selection	Pipeline Depth	Frequency (MHz)		Combinational Area (μm^2)		Non-comb. Area (μm^2)		End-to-end Latency (ns)	
		Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even
16-to-8	7	1,449	3,030	7,788	5,816	10,953	9,860	2.38	2.31
16-to-8	4	2,000	1,923	11,171	7,916	6,693	6,179	2.00	2.10
16-to-8	3	1,515	1,562	9,252	8,482	4,488	4,328	1.98	1.92
32-to-8	11	2,777	2,702	17,477	14,996	25,697	24,268	3.96	4.07
32-to-8	6	1,818	1,851	30,026	24,047	16,502	15,684	3.30	3.24
32-to-8	4	1,470	1,492	27,086	23,175	12,125	11,373	2.72	2.68
64-to-8	15	2,777	2,702	38,249	31,557	57,829	54,986	5.40	5.55
64-to-8	8	1,754	1,818	65,972	48,038	32,103	32,930	4.56	4.40
64-to-8	5	1,470	1,428	69,731	57,929	20,923	21,640	3.40	3.50
128-to-8	19	2,702	2,702	70,847	82,888	109,827	107,933	7.03	7.03
128-to-8	10	1,754	1,754	135,024	108,498	73,844	68,305	5.70	5.70
128-to-8	7	1,333	1,333	128,380	111,700	45,972	45,229	5.25	5.25
256-to-8	23	2,631	2,702	137,774	126,672	226,955	216,093	8.74	8.51
256-to-8	12	1,724	1,694	267,060	194,382	94,730	112,020	6.96	7.08
256-to-8	8	1,298	1,351	262,126	222,433	88,069	88,545	6.16	5.92

As shown in Tables 4.1 and 4.2, the clock frequency scales fairly well for larger max-set-selection units. The reason is that the frequency depends on the CAE width and the number of CAE blocks between pipeline registers. Increasing the number of inputs for a max-set-selection unit does not directly affect the frequency, although it might complicate the wire routing. By decreasing the pipeline depth for a given max-set-selection unit, combinational area increases due to trading increased area from buffers and larger/faster gates for the higher clock frequency. As expected, non-combinational area from registers decreases for a given max-set-selection unit by decreasing the pipeline depth. Comparing the two types of max-set-selection units with each other, odd-even merge max-set-selection units have higher clock frequencies and lower areas than bitonic units for most designs and configurations.

Designs with one CAE stage between pipeline registers have higher sorting throughput than similar designs with two or three CAE stages between pipeline registers, although they usually have higher latency and total area. Designs with three CAE stages between pipeline registers have the lowest latency and total area in most cases, while they provide the lowest sorting throughput. Designs with two CAE stages between pipeline registers attain a trade-off between latency and throughput. For instance, a pipeline depth of seven at about 1.3 GHz provides the lowest latency and lowest resource resource usage for the 256-to-4 max-set-selection unit, while a pipeline depth of 19 at 2.7 GHz gives the highest throughput.

Tables 4.1 and 4.2 show that, for a given number of outputs and pipeline stages, resource requirements scale linearly, latency scales logarithmically, and the frequency scales fairly well with the number of inputs. These results conform to the theoretical analysis of the proposed max-set-selection units in Chapter 3.4. For a given number of outputs, as the number of inputs increases, units with fewer pipeline stages provide better end-to-end latency. Modular design and intelligent pipelining enable efficient frequency/latency trade-offs even for large sorting units.

4.2 FPGA Implementation

We synthesized our proposed designs to a Virtex-5 TX240T FPGA (speed grade -2) using Xilinx Synthesis Technology 11.3 SP3. For all results, the data width (i.e, the CAE width) is set to ten unsigned bits, although the parameterized data width can easily be changed. Table 4.3 shows post-place-and-route results for 2^n-to-4 bitonic max-set-selection units. The designer can specify the pipeline depth of the max-set-selection units, which is given in the second column of Table 4.3. We report results for three different pipeline structures for each 2^n-to-4 bitonic max-set-selection unit: one CAE stage between pipeline registers, two CAE stages between pipeline registers, and three CAE stages between pipeline registers. The last column of Table 4.3 lists the end-to-end latency for each design. All max-set-selection units can achieve high frequencies due to the regular pipelined structure of the designs. For the 256-to-4 max-set-selection unit, a pipeline depth of 7 at 94 MHz provides the lowest latency, while a pipeline depth of 19 at 200 MHz gives the highest throughput.

Table 4.3 shows that, for a given number of outputs and pipeline stages, resource requirements scale linearly, latency scales logarithmically, and the frequency scales fairly well with the number of inputs. These results conform to the theoretical analysis of the proposed max-set-selection units in Chapter 3.4. For a given number of outputs, as the number of inputs increases, units with fewer pipeline stages provide better end-to-end latency. Modular design and intelligent pipelining enable efficient frequency/latency trade-offs on FPGAs even for large sorting units.

4.3 Comparison with Other Approaches

One way to make 2^n-to-2^m sorting units is to create a full 2^n-input network, but use only the top 2^m values, leaving the remaining sorter outputs unused. The synthesis tool can then automatically prune the unused logic. We

Table 4.3: Performance and resource requirements of 2^n-to-4 bitonic max-set-selection units with 10-bit unsigned CAE blocks on an XC5VTX240T-2FF1759 FPGA

Max-set-selection	Pipeline Depth	Freq. (MHz)	Slice LUTs	Slice Regs	End-to-End Latency (ns)
16-to-4	7	277	1,199	760	25.2
16-to-4	4	156	1,198	440	25.6
16-to-4	3	108	1,198	280	27.6
32-to-4	10	250	2,714	1,725	40.0
32-to-4	5	156	2,703	764	32.0
32-to-4	4	108	2,705	604	36.8
64-to-4	13	243	5,727	3,644	53.3
64-to-4	7	149	5,722	2,044	46.9
64-to-4	5	99	5,718	1,244	50.5
128-to-4	16	238	11,832	7,484	67.2
128-to-4	8	138	11,758	3,324	57.6
128-to-4	6	99	11,760	2,524	60.6
256-to-4	19	200	22,382	14,165	95.0
256-to-4	10	126	22,356	7,804	79.0
256-to-4	7	94	22,378	4,644	74.2

compared this method to our custom units and found that our proposed units not only have lower latency, but also require fewer resources. For instance, an automatically optimized 32-to-4 sorting unit is 1.4 times slower and twice as large as our corresponding unit. The advantage of our units over

tool-optimized units soars as the number of inputs increases: our proposed 256-to-4 max-set-selection unit is 1.6 times faster and five times smaller than the corresponding tool-optimized unit.

Parallel sorters can also be implemented in software. A CUDA v3.1 implementation running on an NVIDIA® Tesla™ C2050 GPU at 1.15 GHz requires about 46 µs to sort 256 numbers. GPUs could be a preferred method for implementation of high-throughput sorters when the number of inputs is extremely large due to high cost of FPGA implementation and I/O bandwidth.

4.4 Customized Units Used in the CMS L1 Trigger

In November 2009, the Large Hadron Collider (LHC) [35], built by the European Organization for Nuclear Research (CERN), became the world's highest energy particle accelerator [2]. Experiments using the LHC are intended to address fundamental questions in particle physics and nature. The LHC's Compact Muon Solenoid (CMS) experiment [1] is a high-energy physics detector designed to analyze particle collisions that occur at a rate of roughly one billion collisions per second. The CMS experiment relies on a high-performance system of custom processing elements (collectively known as the CMS trigger) to perform real-time processing of collision data and select the most interesting data for further study. Because of the sheer amount of input data and the rate at which it is generated, the hardware-based CMS trigger is subject to stringent performance requirements [22].

Most of the data generated by collisions in the LHC contains worthless information that corresponds to low-energy particle collisions. This worthless data should be discarded by the CMS trigger in order to process the useful data from high-energy collisions. Therefore, high-performance sorting and max-set-selection units are important components in the CMS trigger for

determining which data to keep and which to discard. The need to modify the CMS trigger design as the LHC system is upgraded, the low-volume cost advantages of FPGAs, and a desire for a flexible and adaptable system all point toward the use of FPGAs as an attractive hardware solution for sorting and max-set-selection units in the CMS trigger. The CMS trigger requires high-bandwidth circuits to process large amounts of data in real-time. By utilizing a sufficient number of high-speed serial transceivers and an efficient communication scheduling methodology, we are able to provide appropriate amounts of data each clock cycler to large sorting units.

Sorting and max-set-selection units are key components throughout the CMS trigger. For example, in the CMS Calorimeter trigger, sorting and max-set-selection units are used to identify important particle interactions that correspond to high-energy collisions. The objects to sort correspond to physics objects (particles and jets) that are ranked by their energy. Energy values are represented using unsigned numbers with six to ten bits. However, there is other information, such as the object position and type that are not used in energy sorting but must be associated with the object's energy. Thus, the sorting is performed by comparing only energy bits and the other bits are passed along with the energy bits. Furthermore, the number of max-set-selection or sorting unit outputs is typically much lower than the number of inputs. For most applications, the sorting unit outputs only the top four objects. For example, the CMS Global Calorimeter trigger (GCT) [52, 54, 55] sorts the candidate electrons, photons, taus, and jets and forwards the four most energetic objects of each type to the Global trigger. Based on the values of these sorted objects and other factors, a decision is made to keep or discard data produced by the CMS Experiment during a specified time interval.

There are several different types of sorting and max-set-selection units utilized in the CMS trigger. The requirements for these units may change over time to cope with changes in the energy of LHC particle collisions and

new physics requirements. Our generalized methodology enables designers to configure sorting and max-set-selection units in terms of the number of inputs and outputs, the width of the input and outputs, the pipeline depth, and even resource usage, thereby achieving suitable high-speed sorting or max-set-selection units for given design constraints.

In the CMS trigger, different sorting and max-set-selection units are used to identify important particle interactions that correspond to high-energy collisions. The inputs to sort correspond to physics objects that are ranked by their energy. Energy values are represented using unsigned numbers with six to ten bits. However, there is other information, such as the object position and type, that is not used in energy sorting but must be associated with the object's energy. Thus, the sorting is performed by comparing only energy bits and the other bits are passed along with the energy bits. In most cases, the sorting unit outputs only the top four objects. For example, the CMS L1 trigger [52, 54, 55] sorts the candidate objects and forwards the four most energetic objects of each type to the global trigger [54]. Based on the values of these sorted objects and other factors, a decision is made to keep or discard data produced by CMS experiments during a specified time interval.

LHC upgrades and physics experiment changes may alter demands on hardware sorting and max-set-selection units over time. Our generalized methodology enables designers to create these units based on the needed input and output width and quantity, throughput, latency, and even resource usage.

The need to modify the CMS L1 trigger design as the LHC system is upgraded, the low-volume cost advantages of FPGAs, and a desire for a flexible and adaptable system all point toward the use of FPGAs as an attractive hardware solution for data processing in the CMS trigger. The CMS trigger requires high-bandwidth circuits to process large amounts of data in realtime and with low latency, including sorting and max-set-selection

Table 4.4: Performance and resource requirements of customized 2^n-to-4 max-set-selection units with 10-bit unsigned energy vectors on an XC5VTX240T-2FF1759 FPGA. Each of the 2^n inputs has an associated n-bit position vector (index).

Max-Set-Selection	Pipeline Depth	Freq. (MHz)	Slice LUTs	Slice Regs	End-to-End Latency (ns)
16-to-4	4	172	1,276	616	23.2
32-to-4	5	149	2,989	1,144	33.5
64-to-4	7	149	6,754	3,268	46.9
128-to-4	8	156	14,374	5,648	51.2
256-to-4	10	147	28,905	14,044	68.0

operations. The FPGA's high-speed serial transceivers provide new input data to these sorting and max-set-selection units every few clock cycles [22].

Using our Verilog model, we generate 2^n-to-4 bitonic max-set-selection units for the CMS trigger with 10-bit unsigned energy vectors, n-bit unsigned position vectors, and pipeline registers after every other CAE stage. Table 4.4 presents the performance and resource requirements of different 2^n-to-4 max-set-selection units for the CMS trigger. Parameterized HDL, regular building blocks, and hierarchical design techniques allow us to quickly develop fast max-set-selection-units with desired specifications. Due to the regularity in our design methodology, increasing the number of inputs does not significantly impact the frequency of the max-set-selection units, making this approach a good candidate for large, high-throughput max-set-selection and sorting units on FPGAs. The 256-to-4 max-set-selection unit in Table 4.4 occupies about 25% of the total slices available on the Virtex-5 TX240T FPGA, allowing designers to implement even larger max-set-selection units or other functionality on the same FPGA.

5 ITERATIVE MAX-SET-SELECTION UNITS

Parallel sorting and max-set-selection units that operate on large blocks of data may receive considerable amounts of input data. Implementing large max-set-selection and partial sorting units in a fully parallel manner requires high I/O bandwidth and area. In addition, for a fixed number of outputs, the resource requirements of these units increase linearly with the number of input values. Thus, fully parallel sorting units may not be practical in large data sorting applications. In cases in which I/O bandwidth or area is limited and latency requirements are not as stringent, a small max-set-selection unit can be employed using an iterative process to obtain the largest values from a given input data set. This iterative approach is particularly well suited to systems in which only a portion of the total data arrives to the max-set-selection unit each cycle and the sorting throughput requirements are not too high. Furthermore, such iterative max-selection units can provide throughput, latency, and resource requirement trade-offs. Max-set-selection and partial sorting units in applications such as HEP and video processing often need to get data from different sources over multiple cycles. Thus, our proposed iterative max-set-selection units, which take as inputs new input data and the largest results from previous iterations, are area-efficient designs for these types of applications.

There has previously been successful research on iterative sorting. Huang *et al.* describe an iterative sorting method that assumes all elements to be sorted are in the device memory and sorts the elements in place [26]. Their memory-based approach does not work efficiently on streaming data that arrives over multiple cycles. Zhang and Zheng [59] implement another iterative sorting method that uses systolic arrays to sort data in memory. Although their approach scales well, it requires special memory hardware. Olariu *et al.* [41] present a hardware algorithm for sorting N values by repeatedly using a fixed-size P-input sorting network that processes P values

each cycle. They show that their algorithm achieves optimal overall performance of $\Theta(\frac{N \times \log_2 N}{P \times \log_2 P})$ provided the P-input sorting network has a depth of $O(\log_2^2 P)$ such as bitonic sorting networks. In this thesis, we instead focus on iterative max-set-selection units. The main differences between our work and previous research include (1) our designs are optimized for the case in which only M outputs from N inputs are needed, (2) our designs avoid using additional storage or intermediate memory blocks by receiving the appropriate number of input values each cycle, and (3) our designs iteratively utilize max-set-selection units, rather than complete sorting units, which leads to improved area and latency.

As shown in Fig. 5.1, our proposed iterative max-set-selection units utilize R-to-M bitonic or odd-even merge max-set-selection units of varying pipeline depths as functional cores. Each design has a finite state machine (FSM) that manages three sequential phases of the execution pipeline: warm-up, steady-state, and completion. The warm-up phase occurs when the first P input values arrive and begin to propagate through the core max-set-selection unit's pipeline, but before any intermediate results are generated. When the core max-set-selection unit outputs data from the first set of input elements, the steady-state phase begins. During each cycle of the steady-state phase, a set of P new input values arrives at the inputs and a set of M intermediate result elements are produced and then immediately consumed by the core max-set-selection unit, where $R = P+M$. In this phase, the intermediate results are fed back into the core max-set-selection unit with the new input values to be sorted. Once all the inputs have been received and applied to the core max-set-selection unit, the completion phase begins, in which intermediate result values are stored at the inputs of the sorting unit as the core max-set-selection unit produces them. Once R values are stored, they are sent to the core max-set-selection unit. This process is completed with a final max-set-selection run with R or fewer remaining valid values, resulting in the final M outputs.

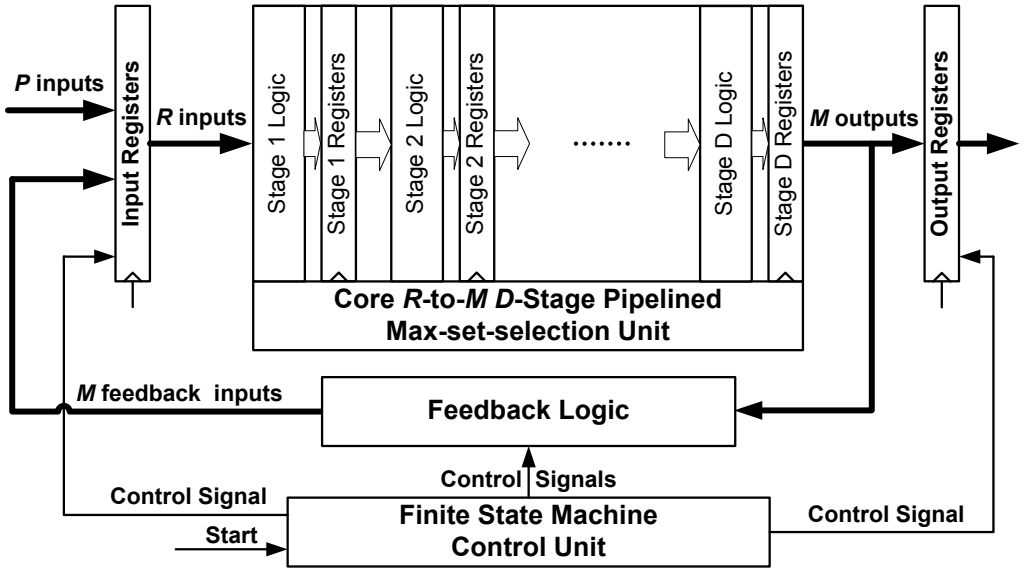

Figure 5.1: Iterative max-set-selection unit.

5.1 Discussion

As shown in Fig. 5.1, we use $R = P+M$ input registers to buffer P input values and M intermediate result values. Removing that level of registers decreases the total number of cycles (latency) to generate the final result. However, it also decreases the overall frequency of the design by adding the delay from the feedback logic into the critical path of the first stage of the core max-set-selection unit. Thus, our designs use input registers to increase the frequency.

Including input registers, the latency of our iterative max-set-selection designs in terms of clock cycles is bounded by

$$\text{Latency} = \lceil N/P \rceil + \left\lceil (D+1)^2 \times M/(P+M) \right\rceil + D + 1 \qquad (5.1)$$

where P is the number of new input values received each cycle by the

core max-set-selection unit, N is the total number of input values, D is the pipeline depth of the core max-set-selection unit, M is the number of outputs from the core max-set-selection unit, and $\lceil \cdot \rceil$ denotes the ceiling operation. It is important to note that N, P, and M define the problem. The first term of the equation accounts for the cycles required to receive all the inputs, both during the warm-up phase and the steady-state phase. The remaining terms describe the bound on how many cycles the completion phase takes.

As an example, a design with a seven-stage, 16-to-4 max-set-selection core unit is used to iteratively perform max-set-selection on 256 inputs. The design parameters are $\mathsf{R} = 16$, $\mathsf{M} = 4$, $\mathsf{N} = 256$, and $\mathsf{D} = 7$, which implies $\mathsf{P} = 12$, since $\mathsf{R} = \mathsf{P}+\mathsf{M}$. The warm-up phase lasts seven cycles in which 12 new inputs are applied each cycle. At the end of the warm-up phase, the input registers and seven pipeline stages each have a set of 12 values being sorted to find the largest four. The steady-state phase begins when the first input set's four largest values are reapplied to the inputs of the core unit along with another 12 completely new inputs. This phase lasts 15 cycles until all 256 inputs are received. The completion phase now forms new sets of 16 values by concatenating (over four cycles) the first next four intermediate output sets and reapplying them to the input of the core unit. The second next four intermediate outputs sets, which were originally in the input registers and first three stages of the pipeline, also form another new set of 16 values in four cycles. After another eight cycles, two output sets of the previous concatenations are reapplied to the core unit. The four largest values from the entire 256 inputs are ready after eight more cycles, for a total of 24 cycles in the completion phase and 46 cycles for the entire process.

An iterative max-selection unit performs a significant percentage of the work as it is receiving input values. More overlapping of computation with data reception (during the warm-up and steady-state phases) occurs by

increasing the total number of input values. Hence, for a large data stream, such as $\mathsf{N} = 8192$, the latency is dominated by the N/P term in Equation (5.1).

5.1.1 Comparison with Parallel Max-set-selection Units

Equation (5.1) demonstrates that the latency of the iterative max-set-selection units scales linearly with the total number of inputs to sort (N), rather than $\mathsf{O}(\log_2^2 \mathsf{N})$, as with parallel max-set-selection units. However, in situations in which P is significantly smaller than N (e.g., I/O bandwidth-limited situations), a parallel max-set-selection unit, which operates on all the input values at one time, must first buffer the input values as they arrive, and then propagate the values through the pipeline only once all input values are present. Moreover, the number of logic inputs of a parallel max-set-selection unit (R) should be as large as the number of data values to sort (N). This imposes a huge impact on the area, since parallel max-set-selection units have a hardware complexity of $\mathsf{O}(\mathsf{N} \times (\log_2^2 \mathsf{M}))$. When the area of a parallel max-set-selection unit can be tolerated, a parallel max-set-selection unit achieves higher throughput and lower latency than an iterative max-set-selection unit.

5.1.2 Iterative Partial Sorting Units

An iterative max-set-selection unit can easily be converted to an iterative partial sorting unit by augmenting the iterative unit with an M-input sorting unit to sort the final M values. This modification increases the latency by $\mathsf{O}(\log_2^2 \mathsf{M})$, if a bitonic or odd-even merge sorting unit is used.

5.2 Results

To illustrate the potential trade-offs that can be made with iterative max-set-selection-units, 18 different iterative max-set-selection units are designed to find the four largest values from a stream of input values. Table 5.1 summarizes the latency and resource requirements of each of the iterative max-set-selection units when they are used to find the four largest values from 256 10-bit input values. The units are synthesized using the TSMC 65-nm standard cell library. These designs use our proposed pipelined 8-to-4, 16-to-4, and 32-to-4 max-set-selection units, described in Chapter 3, as functional cores for the iterative units. By a simple modification to the FSM, our iterative technique can be employed to find the M largest values from N input values using an R-to-M max-set-selection unit as a functional core for any integer values of N, R, and M for which $\mathsf{N} > \mathsf{R} > \mathsf{M} \geqslant 1$ and $\mathsf{R} = \mathsf{P} + \mathsf{M}$.

The area benefit of an iterative max-set-selection over a parallel max-set-selection unit increases linearly with a linear increase in the total number of inputs. For example, in the case of a data stream of $\mathsf{N} = 256$ inputs and where only $\mathsf{P} = 12$ new inputs can be received each cycle, an iterative bitonic max-set-selection unit using a simple 16-to-4 max selection unit with a pipeline depth of four has a latency of 17.64 ns, whereas a parallel bitonic 256-to-4 max-selection unit with a pipeline depth of seven has a latency of 5.39 ns (for the lowest latency 256-to-4 bitonic unit) as indicated in Tables 4.1 and 5.1. However, the iterative max-set-selection unit is more than 13 times smaller than the corresponding parallel unit. Thus, at a small fraction of the resource requirements, the iterative max-set-selection unit could provide reasonable latency and throughput for certain target applications.

Table 5.1: Performance and resource requirements of iterative max-set-selection units used to find the four largest data values from N = 256 data inputs with 10-bit unsigned CAE blocks using a TSMC 65-nm standard-cell library

Core Max-set-selection (R-to-M)	Pipeline Depth (D)	# of New Data Elements per Cycle (P)	Total Cycles	Frequency (MHz)		Combinational Area (μm^2)		Non-comb. Area (μm^2)		End-to-end Latency (ns)	
				Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even	Bitonic	Odd-even
8-to-4	4	4	81	2,702	2,857	3,013	3,222	4,386	4,249	29.97	28.35
8-to-4	2	4	72	1,923	2,000	3,640	3,069	2,567	2,429	37.44	36.00
8-to-4	1	4	68	1,408	1,587	4,837	4,306	1,756	1,666	48.28	42.84
16-to-4	7	12	46	2,632	2,439	6,289	5,590	11,119	9,098	17.48	18.86
16-to-4	4	12	36	2,040	2,083	9,561	8,264	6,398	6,160	17.64	17.28
16-to-4	3	12	30	1,724	1,724	10,801	10,219	6,336	6,130	17.40	17.40
32-to-4	10	28	39	2,174	2,083	10,197	8,191	19,849	17,541	17.94	18.72
32-to-4	5	28	21	1,388	1,388	12,148	11,371	11,339	11,141	15.12	15.12
32-to-4	4	28	19	1,052	1,051	12,352	10,152	9,303	9,467	18.05	18.05

6 RELATED RESEARCH

Sorting algorithms for shared-memory multi-processor systems and VLSI implementations have been investigated during the past 40 years. Linear array structures and sorting networks are two types of architectures that have been widely used for hardware implementations. Different linear sorting units that use insertion techniques have been proposed and implemented as FPGA and VLSI designs [3, 5, 16, 29, 39, 42, 43, 56, 57]. Although they are simple to implement and their area complexity is reasonable, linear array structures are not able to process blocks of data in parallel, resulting in low throughput designs [42]. However, linear array structures are good candidates for sorting streaming or serial data when only one new element is sent to or one sorted element is retrieved from the sorting unit per clock cycle. Linear sorters have a time complexity in the range $\mathsf{O}(\mathsf{b}(\mathsf{N}))$ to $\mathsf{O}(\mathsf{N})$, where b is the bit-width of input values. Compared to sorting networks, the higher time complexity of linear sorters hinders their use in fast, high-throughput sorting applications.

On the other hand, sorting networks, which use multiple levels of parallel CAE blocks to rearrange data, are suitable architectures for sorting huge amounts of parallel data. Moreover, customizable pipelined sorting networks can meet the requirements of high-throughput applications. Hardware-implemented sorting networks have a time complexity of $\mathsf{O}(\log_2^2 \mathsf{N})$ which make them high-speed architectures for large values of N. While theoretical time-optimal $\mathsf{O}(\log_2 \mathsf{N})$ sorting networks have also been proposed [6, 36, 44], they cannot be implemented in hardware because of their large hidden constants in O-notation [34].

6.1 Sorting Networks

From complex cube and mesh array structures to linear array structures, and from theoretical log-depth algorithms to practical linear and $\log^2$-depth algorithms, Batcher's compare-exchange sorting networks are of importance for hardware implementations because of their ease of VLSI realization. Since the advent of Batcher's sorting networks, many sorting schemes have evolved from his bitonic and odd-even merge algorithms for shared-memory systems and VLSI hardware [9, 34, 53]. Previous research on sorting networks falls into two main categories: sorting algorithms and sorting architectures.

Extensive research has been performed to optimize parallel sorting algorithms under various implementation assumptions and for different applications. Herruzo *et al.* [24] propose a novel odd-even merge sorting algorithm based on a divide-and-conquer strategy for shared-memory multi-processor systems. Ionescu *et al.* [27] propose a parallel bitonic sorting algorithm for coarse-grain parallel machines to optimize communication steps and local computations. Agrawal [4] presents a scheme to design arbitrary-sized bitonic sorting networks. He shows that his method can efficiently be used in the design of asynchronous transfer mode (ATM) switches. Kuo *et al.* [32] introduce a modified odd-even merge sorting network for an arbitrary number of inputs. Their modular approach can be used to implement custom sorting units in hardware.

Significant research has also investigated sorting architecture designs for VLSI and FPGA implementation. Latency, throughput, scalability, and resource requirements are the main factors considered for these hardware implementations. Ratnayake *et al.*[49] present an FPGA-based implementation of a modified counting sort algorithm that is used to sort large amounts of data. Their design is composed of memory units to hold data, processing nodes to perform sorting, an address generator to provide addresses for the different memory units, and a control logic to transfer data between the memory units and processing nodes. Layer *et al.* [33] study the hardware

implementation of an iterative sorting unit that provides trade-offs between data throughput and area, and they present a pipelined architecture that utilizes multi-level bitonic sorting networks on FPGAs. They provide a cost function for different sorting networks based on their architecture. Jae-Dong *et al.* [34] propose a novel recirculating bitonic sorting network made up of a level of CAE blocks followed by an Ω-network of $\log_2 \mathsf{N} - 1$ switch levels. The purpose of the recirculating network is to reduce the area complexity of the original bitonic sorting networks to $\mathsf{O}(\mathsf{N} \times \log \mathsf{N})$. Although the proposed network theoretically has the same time complexity as the original bitonic sorting networks, the latency of the switches may degrade the performance of the sorter. The interested reader may refer to [9, 53] for a more detailed survey of hardware sorting algorithms and architectures.

6.2 Partial Sorting and Max-set-selection Units

In some cases, partial sorting units are needed to find the M largest (or smallest) numbers from a set of N numbers in sorted order, where $\mathsf{M} < \mathsf{N}$. Max-set-selection is a related operation that outputs the M largest numbers but not necessarily in sorted order. Partial sorting and selection algorithms have previously been addressed by a wealth of software approaches and techniques [30]. However, hardware designs for partial sorters and selection algorithms have barely been discussed in the literature.

Linear sorters, with slight modifications, are capable of performing partial sorting [50]. Perez-Andrade *et al.* [46] propose a linear FIFO-based sorter to partially sort an arbitrary number of input values. Their algorithm discards the smallest sorted element on receiving a new element and finds a position for the new element. Colavita *et al.* [14] propose a VLSI sorting architecture for streaming data. Their regular design consists of several small elementary sorting units, and its area and latency increase linearly with the number of

values to sort. Consequently, their design does not provide a low-latency solution for sorting large amounts of data. Dong *et al.* [17] propose a parallel partial sorting design using internal FPGA blocks to improve the performance of normalized cross-correlation image matching systems. When a large set of data needs to be sorted, their method can be extended to use FPGA memory blocks.

Each of these partial sorters are based on linear arrays, resulting in a time complexity of $\mathsf{O}(\mathsf{N})$. To find/sort the M largest numbers from N numbers, our proposed partial sorters, which are based on sorting networks, have a time complexity of $\mathsf{O}(\log_2 \mathsf{N} \times \log_2 \mathsf{M})$ and area complexity of $\mathsf{O}(\mathsf{N} \times \log_2^2 \mathsf{M})$, compared with the $\mathsf{O}(\log_2^2 \mathsf{N})$ time complexity and $\mathsf{O}(\mathsf{N} \times \log_2^2 \mathsf{N})$ area complexity of the original bitonic and odd-even merge sorting networks. Our sorters also have better latency, frequency, and throughput than the partial sorters presented above, but our sorters have higher resource requirements as summarized in Table 6.1.

Table 6.1: Complexity of sorting algorithms (N: Total number of inputs, M: Number of outputs, P: Number of new elements per cycle)

Algorithm	Latency	Area	Throughput
Complete sorting network ($N = M = P$) [8]	$O((\log N)^2)$	$O(N \times (\log N)^2)$	$O(N)$
Partial sorting network ($N = P$) *	$O(\log N \times \log M)$	$O(N \times (\log M)^2)$	$O(N)$
Iterative complete sorting network ($N = M$) [41]	$\Theta(\frac{N \times \log N}{P \times \log P})$	$O(P \times (\log P)^2)$ **	$O(P)$
Iterative partial sorting network *	$O(N/P)$	$O((P + M) \times (\log(P + M))^2)$	$O(P)$
Linear complete sorter ($N = M$) [5, 39]	$O(N)$	$O(N)$	$O(1)$
Linear partial sorter [46]	$O(N)$	$O(M)$	$O(1)$

* Our proposed sorting networks.
** It also requires N memory words.

7 CONCLUSIONS

The thesis has presented the design and implementation of flexible, low-latency, high-throughput N-to-M sorting and max-set-selection units and discussed the structure, performance and resource requirements of these units. In this thesis, we propose modular techniques for designing N-to-M sorting and max-set-selection units based on the Batcher's bitonic and odd-even merge sorting algorithms. We present new regular bitonic merging units that are used to construct efficient sorting and max-set-selection units. Although built from Batcher's merging units, our proposed parallel designs modify the original units to obtain efficient max-set-selection and partial sorting units, reducing time and area complexities of the original algorithm to $O(\log_2 N \times \log_2 M)$ and $O(N \times \log_2^2 M)$, respectively. The analysis performed shows that our designs have lower latency and area than previous designs. For instance, a 256-to-4 max-set-selection unit is more than two times faster and five times smaller than the corresponding 256-input complete sorting network.

We employ a modular design methodology that allows our units to be readily utilized in applications with different requirements. Our units meet stringent latency and throughput constraints, are suitable for a wide range of applications, and give designers the flexibility to easily change the sorter architecture. Moreover, our designs can be applied to two's complement or floating-point numbers by simply changing the comparators used in the compare-and-exchange blocks.

Parallel max-set-selection units have high I/O bandwidth and resource requirements. To reduce I/O bandwidth and area, we propose an iterative max-set-selecting method that receives P new input values per cycle. Our iterative design reuses a small max-set-selection unit over a number of iterations to generate the outputs. The proposed iterative units, which have time and area complexities of $O(N/P)$ and $O((P+M) \times \log_2^2(P+M))$, have

much lower resource and I/O requirements than the corresponding parallel units. The iterative max-set-selection units target applications that require selection units with moderate latency and throughput, but need low area and I/O.

REFERENCES

[1] 1998. *CMS: The story of the universe. particle and forces. CERN and LHC. the compact muon solenoid*. Geneva: CERN.

[2] 2009. *LHC sets new world record*. CERN Press Release.

[3] Afghahi, M. 1991. A 512 16-b bit-serial sorter chip. *IEEE J. of Solid-State Circuits* 26(10):1452–1457.

[4] Agrawal, J. P. 1996. Arbitrary size bitonic (ASB) sorters and their applications in broadband ATM switching. In *Proc. IEEE intl. conf. on computers and communications*, 454–458.

[5] Ahn, B., and J. M. Murray. 1989. A pipelined, expandable VLSI sorting engine implemented in CMOS technology. In *Proc. ieee intl. symp. on circuits and systems*, 134–137.

[6] Ajtai, M., J. Komlós, and E. Szemerédi. 1983. An O(n log n) sorting network. In *Proc. ann. acm symp. on theory of computing*, 1–9.

[7] Azuma, Shinsuke, Takao Sakuma, Tetsuya Takeo, Takaaki Ando, and Kenji Shirai. 2000. Diaprism hardware sorter - sort a million records within a second.

[8] Batcher, K. E. 1968. Sorting networks and their applications. In *Afips proc. spring joint computer conference*, 307–314. ACM.

[9] Bitton, Dina, David J. DeWitt, David K. Hsaio, and Jaishankar Menon. 1984. A taxonomy of parallel sorting. *ACM Comput. Surv.* 16(3): 287–318.

[10] Brajovic, Vladimir, and Takeo Kanade. 1999. A VLSI sorting image sensor: Global massively parallel intensity-to-time processing for low-

latency, adaptive vision. *IEEE Trans. on Robotics and Automation* 15(1):67–75.

[11] Chakrabarti, Chaitali. 1993. Sorting network based architectures for median filters. *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing* 40(11):723–727.

[12] Chakrabarti, Chaitali, and Li-Yu Wang. 1994. Novel sorting network-based architectures for rank order filters. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 2(4):502–507.

[13] Chien, M.V., and A. Yavuz Oruc. 1994. Adaptive binary sorting schemes and associated interconnection networks. *IEEE Trans. on Parallel and Distributed Systems* 5(6):561–572.

[14] Colavita, A., A. Cicuttin, F. Fratnik, and G. Capello. 2003. SORTCHIP: A VLSI implementation of a hardware algorithm for continuous data sorting. *IEEE J. Solid-State Circuits* 38(6):1076–1079.

[15] Colavita, Alberto, Enzo Mumolo, and Gabriele Capello. 1997. A novel sorting algorithm and its application to a gamma-ray telescope asynchronous data acquisition system. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 394(3):374–380.

[16] Demirci, T., I. Hatirnaz, and Y. Leblebici. 2003. Full-custom CMOS realization of a high-performance binary sorting engine with linear area-time complexity. In *Proc. intl. symp. on circuits and systems*, vol. 5, V–453–V–456.

[17] Dong, Sheng-Nan, Xiao-Tao Wang, and Xing-Bo Wang. 2009. A novel high-speed parallel scheme for data sorting algorithm based on FPGA. In *Intl. cong. on image and signal processing*, 1–4.

[18] Farmahini-Farahani, A., A. Gregerson, M. Schulte, and K. Compton. 2011. Modular high-Â-throughput and low-Â-latency sorting units for FPGAs in the Large Hadron Collider. In *Proc. ieee intl. symp. on application specific processors*, 38–45.

[19] Govindaraju, Naga, Jim Gray, Ritesh Kumar, and Dinesh Manocha. 2006. GPUTeraSort: high performance graphics co-processor sorting for large database management. In *Proc. of the ACM SIGMOD intl. conf. on management of data*, 325–336.

[20] Graefe, Goetz. 2006. Implementing sorting in database systems. *ACM Comput. Surv.* 38.

[21] Grama, Ananth, George Karypis, Vipin Kumar, and Anshul Gupta. 2003. *Introduction to parallel computing*. 2nd ed. New York: Addison Wesley.

[22] Gregerson, Anthony, Amin Farmahini-Farahani, William Plishker, Zaipeng Xie, Katherine Compton, Shuvra Bhattacharyya, and Michael Schulte. 2009. Advances in architectures and tools for FPGAs and their impact on the design of complex systems for particle physics. *Typical Workshop on Electronics for Particle Physics (TWEPP)* 617–626.

[23] Gregerson, Anthony, Michael Schulte, and Katherine Compton. 2010. High-energy physics. In *Handbook of signal processing systems*, 179–211. Springer.

[24] Herruzo, Ezequiel, Guillermo Ruiz, J. Ignacio Benavides, and Oscar Plata. 2007. A new parallel sorting algorithm based on odd-even mergesort. In *Proc. intl. conf. on parallel, distributed and network-based processing*, 18–22.

[25] Hongwei, Xie, and Xue Yafeng. 2008. An improved parallel sorting algorithm for odd sequence. In *Intl. conf. on advanced computer theory and engineering*, 356–360.

[26] Huang, Chun-Yueh, Gwo-Jeng Yu, and Bin-Da Liu. 2001. A hardware design approach for merge-sorting network. In *Proc. ieee intl. symp. on circuits and systems*, vol. 4, 534–537.

[27] Ionescu, M. F., and K. E. Schauser. 1997. Optimizing parallel bitonic sort. Tech. Rep., University of California at Santa Barbara.

[28] Ja'Ja', J., and R. M. Owens. 1984. VLSI sorting with reduced hardware. *IEEE Trans. on Computers* C-33(7):668–671.

[29] Karthik, S., I. de Souza, J. T. Rahmeh, and J. A. Abraham. 1991. Interlock schemes for micropipelines: Application to a self-timed rebound sorter. In *Proc. ieee intl. conf. on computer design: Vlsi in computers and processors*, 393–396.

[30] Knuth, Donald E. 1998. *Art of computer programming, volume 3: Sorting and searching.* 2nd ed. Reading, Mass: Addison-Wesley.

[31] Koch, Dirk, and Jim Torresen. 2011. FPGASort: A high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting. In *Proc. ACM/SIGDA intl. symp. on field programmable gate arrays*, 45–54.

[32] Kuo, Chung J., and Zhi W. Huang. 2001. Modified odd-even merge-sort network for arbitrary number of inputs. In *IEEE intl. conf. on multimedia and expo*, 929–932.

[33] Layer, C., D. Schaupp, and H.-J. Pfleiderer. 2007. Area and throughput aware comparator networks optimization for parallel data processing on FPGA. In *Intl. symp. on circuits and systems*, 405–408.

[34] Lee, Jae-Dong, and K. E. Batcher. 2000. Minimizing communication in the bitonic sort. *IEEE Trans. on Parallel and Distributed Systems* 11(5):459–474.

[35] Lefevre, C. 2008. LHC: The guide.

[36] Leighton, Tom. 1985. Tight bounds on the complexity of parallel sorting. *IEEE Trans. on Computers* C-34(4):344–354.

[37] Liszka, Kathy J., and Kenneth E. Batcher. 1993. A generalized bitonic sorting network. *Proc. Intl. Conf. on Parallel Processing* 105–108.

[38] Marcelino, Rui, Horácio C. Neto, and João M. P. Cardoso. 2008. Sorting units for FPGA-based embedded systems. In *IFIP cong. distributed embedded systems: Design, middleware and resources*, 11–22.

[39] Moore, Simon W., and Brian T. Graham. 1995. Tagged up/down sorter - a hardware priority queue. *The Computer Journal* 38:695–703.

[40] Nakatani, T., S.-T. Huang, B.W. Arden, and S.K. Tripathi. 1989. K-way bitonic sort. *IEEE Trans. on Computers* 38:283–288.

[41] Olariu, S., M. C. Pinotti, and S. Q. Zheng. 2000. An optimal hardware-algorithm for sorting using a fixed-size parallel sorting device. *IEEE Trans. on Computers* 49(12):1310–1324.

[42] Ortiz, J., and D. Andrews. 2010. A configurable high-throughput linear sorter system. In *Proc. ieee intl. symp. on parallel distributed processing (ipdpsw)*, 1–8.

[43] Parhami, B., and Ding-Ming Kwai. 1999. Data-driven control scheme for linear arrays: Application to a stable insertion sorter. *IEEE Trans. on Parallel and Distributed Systems* 10(1):23–28.

[44] Paterson, Mike. 1990. Improved sorting networks with O(log N) depth. *Algorithmica* 5(1):65–92.

[45] Pedroni, V.A., R.P. Jasinski, and R.U. Pedroni. 2010. Panning sorter: A minimal-size architecture for hardware implementation of 2D data sorting coprocessors. In *IEEE asia pacific conf. on circuits and systems*, 923–926.

[46] Perez-Andrade, R., R. Cumplido, F. Del Campo, and C. Feregrino-Uribe. 2008. A versatile linear insertion sorter based on a FIFO scheme. In *Proc. ieee comp. soc. ann. symp. on vlsi*, 357–362.

[47] Pitas, I., and A. N. Venetsanopoulos. 1990. *Nonlinear digital filters: Principles and applications*. Boston, Mass: Kluwer Academic Publishers.

[48] Pok, D.S.K., C.-I.H. Chen, J.J. Schamus, C.T. Montgomery, and J.B.Y. Tsui. 1997. Chip design for monobit receiver. *IEEE Trans. on Microwave Theory and Techniques* 45(12):2283–2295.

[49] Ratnayake, Kumara, and Aishy Amer. 2007. An FPGA architecture of stable-sorting on a large data volume: Application to video signals. In *Proc. ann. conf. on information sciences and systems*, 431–436.

[50] Ribas, L., D. Castells, and J. Carrabina. 2004. A linear sorter core based on a programmable register file. In *Proc. conf. on design of circuits and integrated systems*, 635–640.

[51] Stephens, Donpaul C., Jon C.R. Bennett, and Hui Zhang. 1999. Implementing scheduling algorithms in high-speed networks. *IEEE J. on Selected Areas in Communications* 17:1145–1158.

[52] Taurok, A., H. Bergauer, and M. Padrta. 2001. Implementation and synchronisation of the first level global trigger for the CMS experiment at LHC. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 473(3):243–259.

[53] Thompson, C. D. 1983. The VLSI complexity of sorting. *IEEE Trans. on Computers* C-32(12):1171–1184.

[54] Varela, J. 2002. CMS L1 trigger control system. *CMS Note 2002/033*.

[55] Wulz, Claudia-Elisabeth. 2001. Concept of the first level global trigger for the CMS experiment at LHC. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 473(3):231–242.

[56] Yasuura, H., N. Takagi, and S. Yajima. 1982. The parallel enumeration sorting scheme for VLSI. *IEEE Trans. on Computers* C-31(12):1192–1201.

[57] Yen-Chun, Lin. 1993. On balancing sorting on a linear array. *IEEE Trans. on Parallel and Distributed Systems* 4(5):566–571.

[58] Yun, K.Y., K.W. James, R.H. Fairlie-Cuninghame, S. Chakraborty, and R.L. Cruz. 2000. A self-timed real-time sorting network. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 8(3):356–363.

[59] Zhang, Yanjun, and S. Q. Zheng. 2000. An efficient parallel VLSI sorting architecture. *VLSI Design* 11(2):134–147.