Scalable Binary Sorting Architecture Based on Rank Ordering With Linear Area-Time Complexity

İ. Hatırnaz and Y. Leblebici
Department of Electrical and Computer Engineering
Worcester Polytechnic Institute

Abstract

A new modular architecture is presented for the realization of high-speed binary sorting engines, based on efficient rank ordering. Capacitive Threshold Logic (CTL) gates are utilized for the implementation of the multi-input programmable majority (voting) functions required in the architecture. The overall complexity of the proposed bit-serial architecture increases *linearly* with the number of input vectors to be sorted (window size = m) and with the bit-length of the input vectors (word size = n), and the sorter architecture can be easily expanded to accommodate large vector sets. It is demonstrated that the proposed sorting engine is capable of producing a fully sorted output vector set in (m+n-1) clock cycles, i.e., in linear time.

1 Introduction

The task of sorting an arbitrarily ordered vector set according to magnitude (either from-largest-to-smallest or from-smallest-to-largest) is one of the fundamental operations required in many digital signal processing applications. It is also among the best studied problems in computer science, with a variety of different algorithms developed for this purpose. Many fundamental computer science problems like searching, finding the closest-pair, and frequency distribution etc., become easy to solve once a set of items is sorted (Fig. 1).

Sorting is an expensive operation in terms of area-time complexity; software-based solutions require word-level sorting and can become computationally intensive, while the overall complexity of hardware-based solutions usually increases very rapidly with the size of the input vector set (number of vectors) and with the bit-length of the input vectors [1], [3], [5]. The design of efficient sorting engine architectures is therefore a significant challenge for overcoming the computational bottleneck of the binary sorting problem. A number of recent proposals for the realization of sorting networks rely primarily on median or rank order filters (ROF), yet their capabilities in terms of window size and bit-length are typically limited due to rapidly increasing hardware complexity [2], [5], [6] .

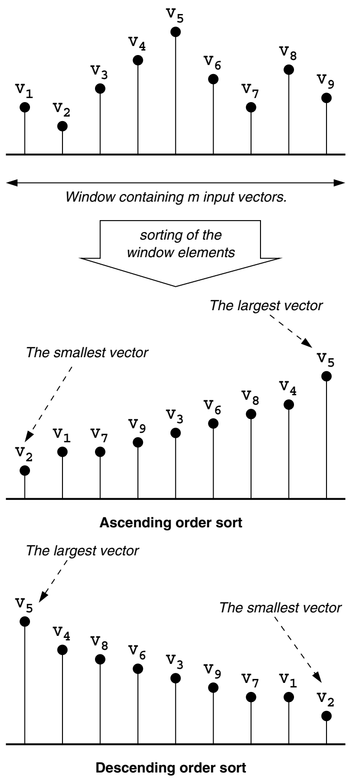

Figure 1: Illustration of the sorting process (1-D).

In this paper, we present a new bit-serial sorting architecture based on rank-ordering. The hardware realization of this architecture results in a compact and fully modular sorting engine architecture that is capable of processing a large number of input vectors in linear time. The overall architecture is completely scalable to accommodate a wide

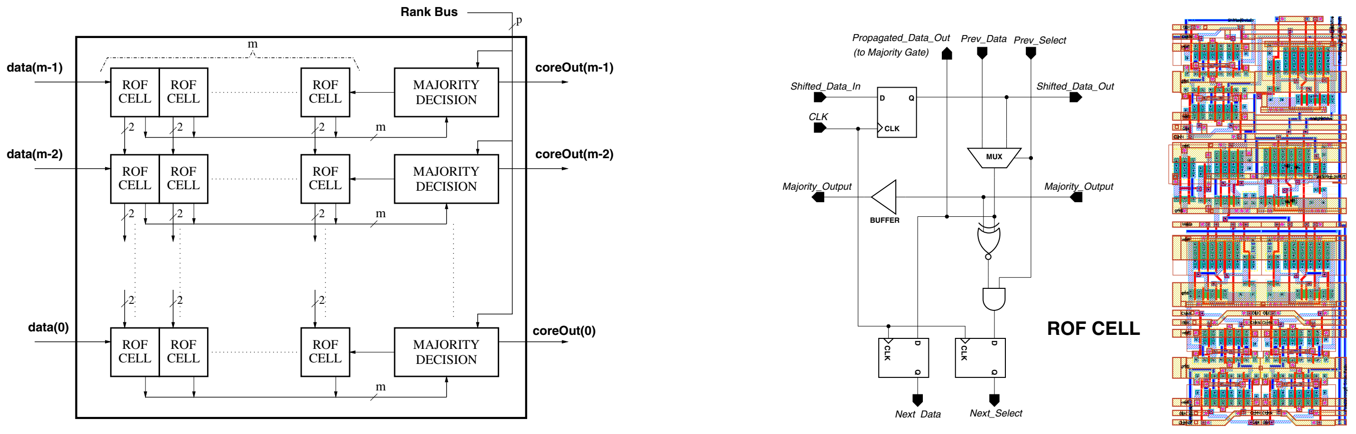

Figure 2: The ROF core as proposed in the modular rank ordering architecture, the gate-level structure of a ROF cell, and the corresponding layout, allowing modular expansion.

range of window sizes and bit-lengths, and the hardware complexity only grows linearly with both of these parameters. The proposed sorter architecture is essentially based on a fully programmable modular ROF design that was presented earlier [7]. In the following, the rank ordering architecture is presented in Section 2. The proposed sorting algorithm and its hardware realization are examined in Section 3 and 4, followed by a summary of results.

2 The Rank Ordering Architecture

A bit-serial algorithm proposed in [5] was chosen as the basis of the programmable rank-order filter architecture implemented in this work. In this algorithm, the problem of finding a rank-order-selection for n-bit long words is reduced to finding "n" rank-order-selections for 1-bit numbers.

The algorithm starts by processing the most significant bits (MSB) of the m=(2N + 1) words in the current window, through an m-input programmable majority gate, to yield the MSB of the desired filter output. This output is then compared with the other MSBs of the window elements. The vectors whose MSB is not equal to the filter output have their MSB propagated down by one position, replacing the less significant bits of the corresponding words.

The bit-serial operation flow of the algorithm described above suggests a simple bit-level pipelined data path architecture, consisting of data modifier-propagator blocks (ROF Cells) to handle fine-grained data selection, and majority decision blocks (majority gate) to determine output bits. The modular architecture consisting of these two major blocks enables fully scalable construction of filter structures of arbitrary window size and bit-length (Figure 2). The bit-length dictates the number of the majority decision gates, whereas the window size determines the number of ROF-cells driving one of these majority gates. This regular structure also forms the basis of the sorting algorithm described in the next section.

The ROF core shown in Figure 2 has (n·m) ROF cells where $m = (2N + 1)$ is the window size and n is the bit-length of the input words (vectors). Thus, it can be seen that the overall circuit complexity increases linearly with maximum window size (m) and with bit-length (n).

A prototype ROF circuit has been designed and fabricated using a 0.8 μm CMOS process, to validate the main operation principles of the ROF architecture. Detailed measurement results of this circuit indicate that the new architecture can accommodate sampling clock rates up to 50 MHz [7]. In this circuit, the programmable majority decision gates are realized using the capacitive threshold logic (CTL) circuit architecture presented earlier [4]. This allows simple implementation of programmable majority gates with up to 63 parallel inputs, using a very small silicon area ($625\mu m$ x $130\mu m$ for 63-bit majority gate). In comparison, a classical realization of the 63-bit majority gate would require an equivalent of 63 6-bit full-adder circuits, arranged in a network of a logic depth of 64 (synthesized from HDL description).

3 The Sorting Algorithm

In the following, we propose a bit-serial sorting algorithm with an input set (window) of "m" n-bit words. The output corresponds to a sequence of the input vectors in a desired rank order. The algorithm starts by processing the MSBs of the "m" input vectors in the current window. Note that each bit-plane can be independently assigned its own rank value which is used to calculate the slice output.

The pseudo-code of the proposed sorting algorithm is given below. The algorithm involves two loops; the outer loop initializes the rank value for the next iteration and checks if the sorting operation is finished, whereas the inner loop does the actual sorting operation by performing parallel instructions on "n" bit-planes.

```
rankof(1):= firstDesiredRank;
do{
  -- Rank initialization
  if (rankof(1) != lastDesiredRank)
    rankof(0):= nextDesiredRank;
  -- Main operation starts
  for all bit-planes
  do{
    if (all_bits(selectedBitplane) are in the core)
       then shift_rotate(selectedBitplane);
    else
       shift(selectedBitplane);
    end if;
    selectedRank := rankof(selectedBitplane);
    outputWordVector(selectedRank, selectedBitplane) :=
     rank_order(selectedBitplane, selectedRank);
    rankof(selectedBitplane) :=
     rankof(selectedBitplane - 1);
  }
}while (rankof(n) != lastDesiredRank)
```

Listing of the proposed sorting algorithm.

The very first step of the algorithm is to set the rank value of the most-significant bit-plane to the first desired rank (*firstDesiredRank*), whose value depends on whether the input vectors are to be sorted in ascending or descending order. For example, if we consider the case of sorting the input vectors in ascending order; at the first iteration of the main operation loop (inner loop), the rank value corresponding to the most-significant bit-plane ($rankof(1)$) has to be set to "*smallestRank*", which results in filtering out the smallest input word. Also, the rank values for the next iterations (*nextDesiredRank*) are determined by the sorting direction, that are stored in a register ($rankof(0)$). If the vectors are to be sorted in ascending order, the value of $rankof(0)$ is increased until the rank value corresponding to the MSB-plane will be equal to the upper rank value (*lastDesiredRank*), which will be the "*largestRank*". So, at each step, the value of $rankof(0)$ is assigned to the MSB slice ($rankof(1)$), where as the rank value of each slice is shifted to one lesser-significant bit-slice. It should also be noted that the algorithm can be used for sorting the input vectors in any desired order. In this case, a look-up table may be used to provide the necessary sequence of rank values to the sorter engine core.

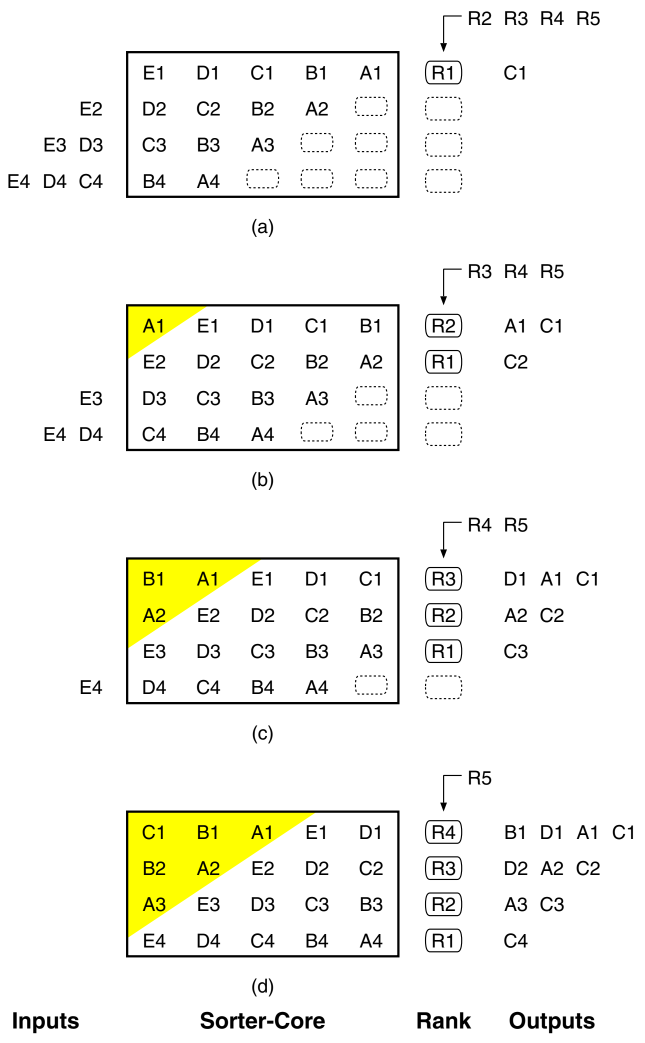

Figure 3: Illustration of a sorting operation on five 4-bit input vectors: (a) The staggered input vectors are shifted into the ROF core, and the first rank (R1) is applied to the MSB plane. (b) The MSB of the first input vector (A1) is rotated, R1 is applied to the next bit-plane, and the new rank R2 is applied to the MSB plane. (c) B1 and A2 are rotated, while R1 is applied to the lesser-significant bit-plane. The rank R2 shifts down by one, while R3 is applied to the MSB plane. (d) Bit circulation continues, while the ranks propagate down the bit-planes in descending order. In this example, the rank ordering of the input vectors is assumed to be: C (largest vector)-A-D-B-E (smallest vector).

The operations contained in the inner loop are performed at the same time on all bit-planes. After the "m" bits in each bit-plane are arranged either by shifting or by shifting and rotating, the corresponding bit-plane output

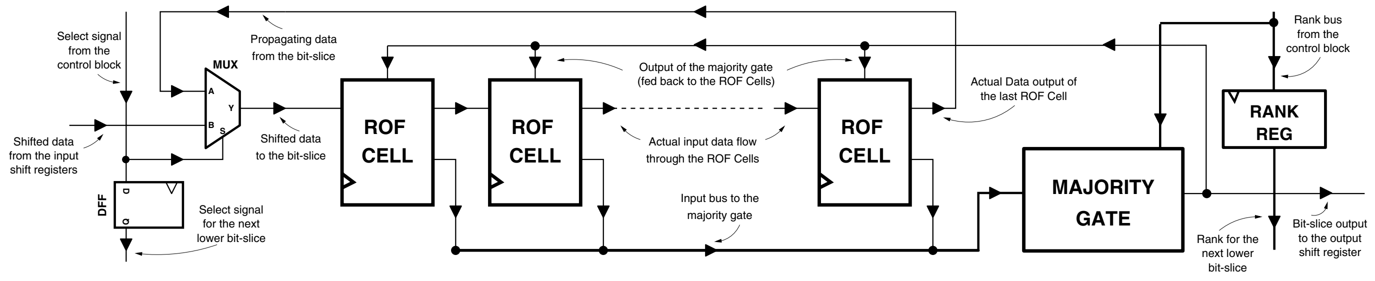
Select signal from the control block
Propagating data from the bit-slice
MUX
A
B
Y
S
Shifted data from the input shift registers
Shifted data to the bit-slice
DFF
D
Q
Select signal for the next lower bit-slice
ROF CELL
ROF CELL
ROF CELL
Output of the majority gate (fed back to the ROF Cells)
Actual input data flow through the ROF Cells
Input bus to the majority gate
Actual Data output of the last ROF Cell
MAJORITY GATE
Rank bus from the control block
RANK REG
Rank for the next lower bit-slice
Bit-slice output to the output shift register

Figure 4: The signal flow of one sorter slice bit-slice, showing how the data bits are circulated.

is calculated by evaluating all of the bits in each bit-plane according to the current rank value ("*rank_order*"). The algorithm is finished after the bit-plane corresponding to the least-significant bits is processed with the last rank value (*lastDesiredRank*).

The operation of the proposed sorting algorithm is illustrated with an example in Fig. 3. Here, five 4-bit vectors (A through E) are being sorted by the ROF core. The first rank (R1) is initialy applied to the MSB plane consisting of the bits A1 through E1. In the next clock cycle, the same rank is used to process the lesser-significant bit-plane (A2 through E2), while a new rank (R2) is being applied to the MSB plane. Also note that the staggered data bits are gradually circulated from the end of the chain to the front, so that each vector in the window can be completely processed. The entire operation requires only (m+n-1) clock cycles after all input vectors are applied. It is important to note that the time-complexity of the sorting operation described above has a linear dependence both with respect to window size (m) and with respect to bit-length (n).

4 Realization of the Sorting Engine

The proposed sorter architecture exploits the fact that the modular ROF core described in Section 2 is capable of generating one output vector per clock cycle, corresponding to the currently selected rank. If the ranking process is repeated on the same set of vectors instead of processing a continuous stream of new vectors, the members of the vector set can be sorted in linear time by simply changing (increasing or decreasing) the rank in each clock cycle. Figure 4 shows the circuit structure and the signal flow of one sorter bit slice that is designed to implement the bit-level operations described above. The multiplexer on the input side is used for accepting the input vectors at the rate of one vector per clock cycle, as well as for circulating (rotating) the data until sorting is completed. The so-called "sorter core" is simply constructed by stacking "n" such bit-slices, as depicted in Fig. 5.

The overall architecture of the sorting engine is shown in Fig. 5. The flow of data through the modular ROF core is being regulated by complementary input and output shift registers, which are used to stagger the individual bit-planes of each input vector to enable bit-level pipelined operation. The control logic is responsible for regulating the data circulation path, and for applying the rank selection signals to the individual bit-planes, in ascending or descending order. The fact that each individual bit-plane is capable of processing a different rank at any given time significantly increases the overall efficiency of this architecture. In a typical sorting run, the control logic simply requests each bit-plane to process a different rank in each clock cycle, either beginning from the maximum rank and descending, or beginning from the minimum rank and ascending.

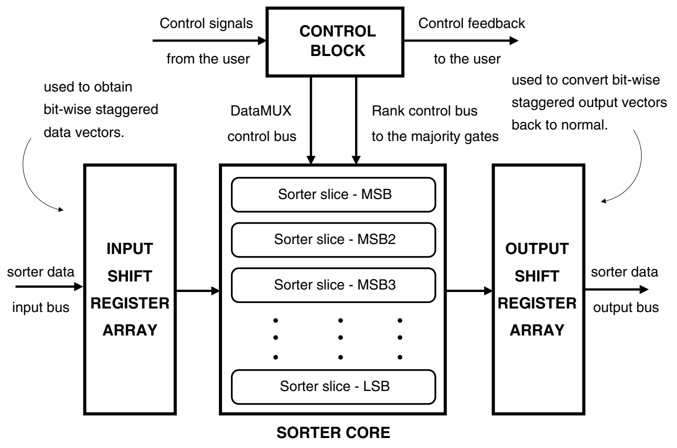
Control signals from the user
CONTROL BLOCK
Control feedback to the user
used to obtain bit-wise staggered data vectors.
DataMUX control bus
Rank control bus to the majority gates
used to convert bit-wise staggered output vectors back to normal.
sorter data input bus
INPUT SHIFT REGISTER ARRAY
Sorter slice - MSB
Sorter slice - MSB2
Sorter slice - MSB3
Sorter slice - LSB
SORTER CORE
OUTPUT SHIFT REGISTER ARRAY
sorter data output bus

Figure 5: Top-level blocks in the architecture of the proposed sorter engine. Note that each slice in the "sorter core" contains m ROF cells, one data MUX and one m-input majority gate.

The proposed architecture has been described with VHDL to verify its operation. Figures 6 and 7 show simulated results of the sorting operation on an arbitrarily

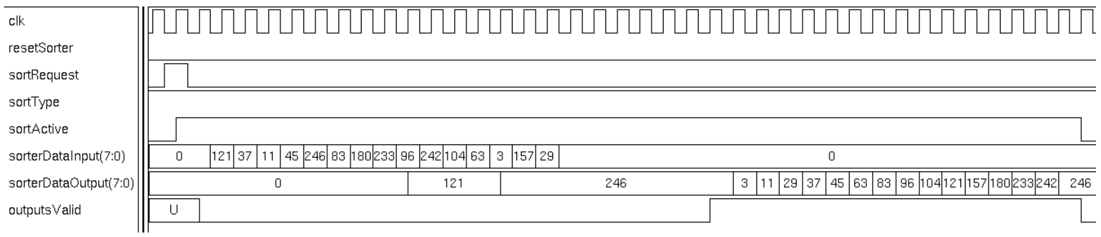

Figure 6: VHDL simulation results of the proposed sorter architecture (m=15, n=8, ascending order).

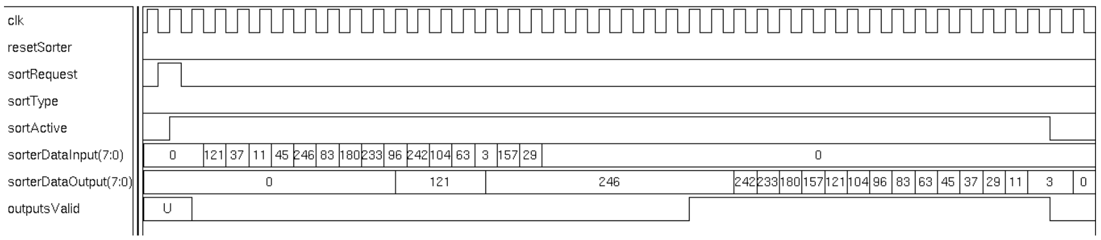

Figure 7: VHDL simulation results of the proposed sorter architecture (m=15, n=8, descending order).

ordered set of 15 vectors (m=15), each with a bit-length of 8 bits (n=8). The user determines how many input vectors are to be sorted ("*actualWindowSize*", not shown in Fig. 6 and Fig. 7) and in which direction the sorting will occur ("*sortType*") and provides these inputs to the sorter block together with a request pulse ("*sortRequest*"). As soon as the request comes, the sorter block produces signal ("*sortActive*") which stays at the logic high level as long as the corresponding set of vectors is processed. It can be seen that the first output vector is generated with a latency of (n-1) clock cycles, after the last vector of the set is entered. The sorter block provides a signal to the user ("*outputsValid*") which goes high right at the last rising edge of the clock before the first vector is ready at the output ("*sortDataOutput*").

The mask layout of a (63x16-bit) sorter block was completed using 0.8 um CMOS technology, to evaluate the area-efficiency of the presented architecture. The entire sorting engine occupies a silicon area of 37.7 sqmm, about 80% of which is dedicated to the ROF cells, and 10% of which is dedicated to majority gates.

5 Summary

In this paper, we present a highly modular architecture for the realization of high-speed binary sorting engines. The architecture consists of (i) a regular "core" array that is completely scalable to accommodate large window sizes and bit-lengths, (ii) input/output shift registers, and (iii) control logic to regulate the bit-level processing of data. It was shown that the complexity of the proposed bit-serial pipelined architecture increases *linearly* with the number of input vectors (m) to be sorted, and with bit-length of the input vectors (n). It was also demonstrated that the proposed sorting engine is capable of producing a fully sorted output vector set in (m+n-1) clock cycles, i.e., in *linear* time.

References

[1] D.S. Richards, "VLSI median filters", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp.145-152, Jan 1990.

[2] W.K. Lam and C.K. Li, "Binary sorter by majority gate", *IEE Electronic Letters*, Vol. 32, July 1996.

[3] P. Wendt et al., "Stack filters", *IEEE Trans. Acoust., Speech, Signal Processing*, pp. 898-911, 1986.

[4] Y. Leblebici, F.K. Gurkaynak, D. Mlynek, "A compact 31-input programmable majority gate based on capacitive threshold logic", *in Proc. ISCAS 1998.*

[5] B.K. Kar, D.K. Pradhan, "A new algorithm for order statistic and sorting", *IEEE Trans. on Signal Processing*, vol. 41, pp.2688-2694, August 1993.

[6] C.C. Lin, C.J. Kuo, "Fast response 2-D rank order algorithm by using max-min sorting network", *Int. Conf. on Image Processing 1996*, Vol. 1, pp. 403-406.

[7] İ. Hatırnaz, F.K. Gurkaynak, Y. Leblebici, "A modular and scalable architecture for the realization of high-speed programmable rank-order filters", *ASIC'99 Proceedings*, pp. 382-386, 1999.