A COMPACT MODULAR ARCHITECTURE FOR HIGH-SPEED BINARY SORTING

İ. Hatırnaz, F. K. Gürkaynak, Y. Leblebici

Worcester Polytechnic Institute
Department of Electrical and Computer Engineering
Worcester, MA 01609-2280

ABSTRACT

A new algorithm and a new modular architecture are 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. Detailed simulations indicate that the sorter structure can operate at sampling clock rates of up to 50 MHz, where the throughput is boosted by fine-grain pipelining. It is demonstrated that the proposed sorting engine is capable of producing a fully sorted output vector set in (m+n-1) clock cycles.

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 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 [5], [1], [3]. 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 [5], [6], [2].

In this work, we present a new bit-serial sorting algorithm based on rank-ordering. The hardware realization of this algorithm results in a compact and fully modular sorting engine architecture that is capable of processing a large number of input vectors in linear time (Fig. 1). The overall architecture is completely scalable to accommodate a wide 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 [9]. In the following, we first present the rank ordering architecture in Section 2, followed by the proposed sorting algorithm and its hardware realization.

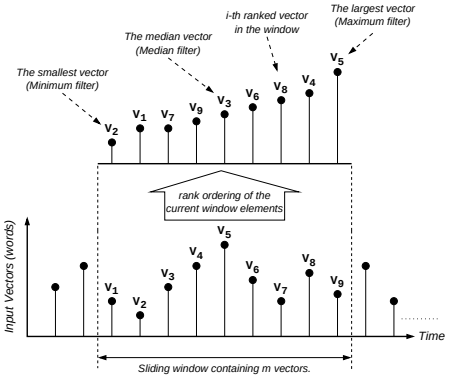

Figure 1: Illustration of the rank-ordering process.

2. THE RANK ORDERING ARCHITECTURE

A bit-serial algorithm first 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 to handle fine-grained data selection, and majority decision blocks to determine output bits.

A programmable rank-order filter of any window size and bit-length can be realized by using the two main blocks described above. 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 results in the ROF core array shown in Fig. 2. 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).

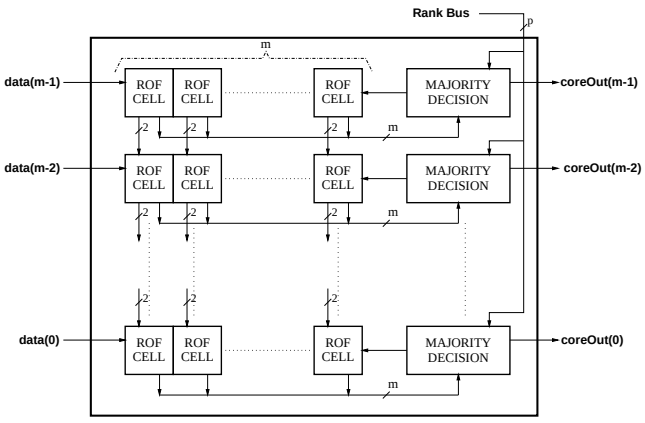

Figure 2: The ROF core as proposed in the ROF architecture.

The signal flow between the ROF cells and the majority gates are also shown in Figure 2. The modular architecture consisting of only two major blocks enables fully scalable construction of filter structures of arbitrary size. It also forms the basis of the sorting algorithm described in the next section.

3. THE SORTING ALGORITHM

The proposed sorting algorithm is a bit-serial algorithm, whose input is a window of "m" n-bit words. The output corresponds to a sequence of the input vectors in a desired rank order. It starts by processing the MSBs of the "m" input vectors in the current window. Each bit-plane has its own rank value which is used to calculate the corresponding 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 check 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_the_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. If sorting is in ascending order, the $rankof(0)$ will be assigned "$rankof(1) + 1$", until the rank value corresponding to the MSB-plane will be equal to the upper rank value (*lastDesiredRank*), which will be the "*largestRank*". 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.

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&rotating, the corresponding bit-plane output 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).

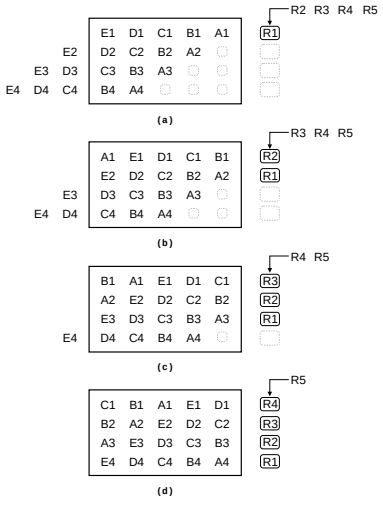

Figure 3: Illustration of a sorting operation on five 4-bit input vectors: (a) The staggered input vectors are shifted into the ROF core, and the first rank (R1) is applied to the MSB plane. (b) The MSB of the first input vector (A1) is rotated, R1 is applied to the next bit-plane, and the new rank R2 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.

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. The overall architecture of the sorting engine is shown in Fig. 4. 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 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 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.

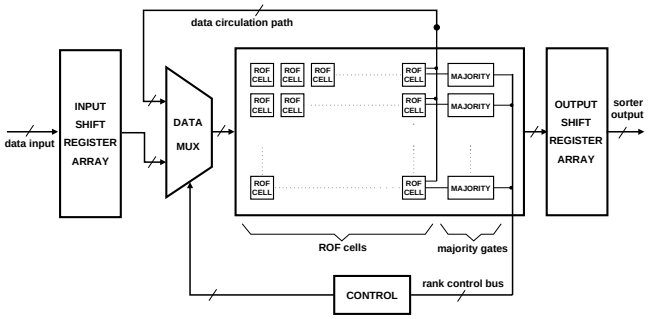

Figure 4: Overall architecture of the proposed sorter engine.

The proposed architecture has been described with VHDL to verify its operation. Fig. 5 shows simulated results of two sorting operations on an arbitrarily ordered set of eight vectors, each with a bit-length of 8 bits. 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.

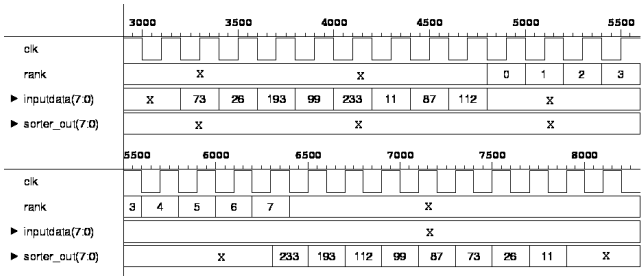

Figure 5: Simulation results of a ranking operation on an arbitrarily ordered set of eight vectors. The input set is being sorted in descending order from maximum (233) to minimum (11) value. The input set can also be sorted in ascending order from minimum to maximum value, simply by changing the rank sequence applied to the majority gates of each bit-plane.

5. CONCLUSION

A modular architecture has been presented for the realization of high-speed binary sorting engines, based on an efficient rank ordering scheme. The overall complexity of the proposed bit-serial architecture increases linearly with the number of input vectors to be sorted and with the bit-length of the input vectors. It was demonstrated that the proposed sorting engine is capable of producing a fully sorted output vector set in (m+n-1) clock cycles.

6. REFERENCES

[1] D.S. Richards, "VLSI median filters", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp.145-152, January, 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. IEEE Int. ASIC Conference 1998*, pp. 281-285.

[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", *International Conference on Image Processing 1996*, Vol. 1, pp. 403-406.

[7] C. Chen, L. Chen, T. Chiueh, J. Hsiao, "An efficient pipelined VLSI implementation of rank order filter", *IS-SIPNN 1994*, Vol. 2, pp. 630-633.

[8] C.L. Lee and C.W. Jen, "Bit-sliced median filter design based on majority gate", *in Proc. Ins. Elec. Eng.-G*, vol 139, pp.63-71, 1992.

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