In-Memory Sorting

For high-performance applications, sorting is implemented in hardware using ASICs or FPGAs.

Parallel nature of hardware-based solutions allows to outperform software-based solutions.

Usual approach for h/w-based sorting is to wire up a network of Compare-and-Swap (CAS) units in a configuration called a Batcher (or bitonic) network.

A bitonic sequence is a sequence of numbers x_0, x_1, $\cdots$, x_{n-1} with following properties:

1. There exists an index i, where 0$\leq i \leq$$n$-1 such that $a_0 \leq a_1 \cdots \leq a_i$ and $a_i \geq a_{i+1} \geq \cdots \geq a_{n-1}$.
   Example: Sequence 1, 2, 4, 7, 6, 0

2. It can be cyclically shift a_i such that (1) is true.
   Example: Sequence 8, 9, 2, 1, 0, 4 : cyclic rotation of 0, 4, 8, 9, 2, 1

Consider the bitonic sequence: x = (21, 20, 14, 10, -6, -4, 0, 1, 2, 18, 19, 30, 31, 25, 23, 22)

It can be cyclically rotated as -4, 0, 1, 2, 18, 19, 30, 31, 25, 23, 22, 21, 20, 14, 10, -6.

The sequence 1, 2, 1, 2 is not bitonic.

Sorting strategy: Bitonic sort

Let S = $<a_0, a_1 \cdots, a_{n-1}>$ be a bitonic sequence

such that $a_0 \leq a_1 \cdots \leq a_{n/2-1}$ and $a_{n/2} \geq a_{n/2+1} \geq \cdots \geq a_{n-1}$.

Consider subsequences

S1 = <min$\{a_0, a_{n/2}\}$, min$\{a_1, a_{n/2+1}\}$, $\cdots$, min$\{a_{n/2-1}$,$a_{n-1}>$ and

S2 = <max$\{a_0, a_{n/2}\}$, max$\{a_1, a_{n/2+1}\}$, $\cdots$, max$\{a_{n/2-1}$,$a_{n-1}>$.

Note that S1 and S2 are both bitonic and each element of S1 is less that every element in S2.

This procedure can be applied recursively on S1 and S2 to get the sorted sequence.
Use Compare-and-Swap (CAS) units

Each CAS block compares two input values and, if required, swaps the values at the output.

Figure 1.8(a) shows the schematic symbol of a CAS block.

Steps

1. Create a bitonic sequence S from an unsorted list Y.
2. Sort S by splitting into two bitonic sequences L(S) and R(S)
   sorting these recursively and then merging.

Let original Y is 19, 2, 72, 3, 18, 57, 600, 100.

The generation of bitonic sequence is shown in Figure 1.7.

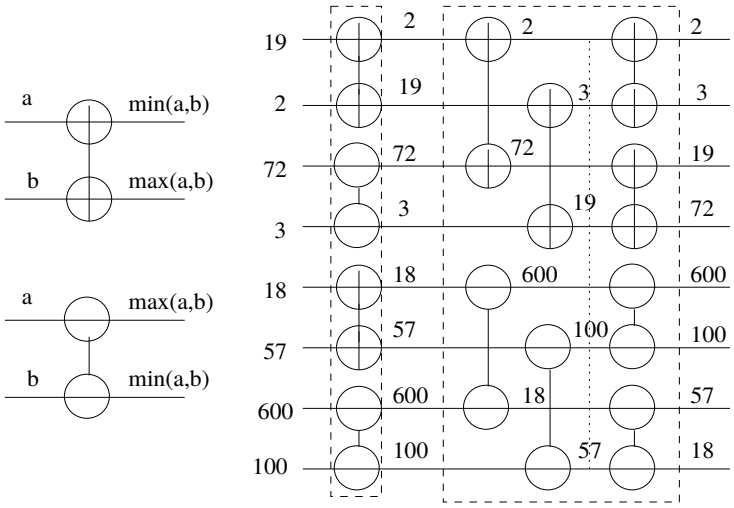


Figure 1.7: Bitonic sequence from unsorted array

Figure 1.8(b) shows CAS network for an 8-input bitonic sorting network, made up of 24 CAS blocks.

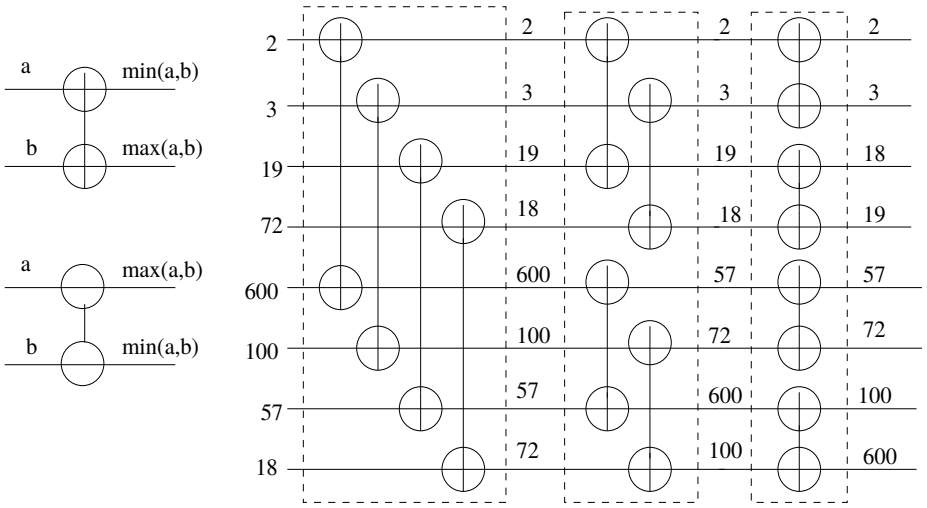


Figure 1.8: CAS block and CAS network for an 8-input bitonic sorting

Implementation cost of a batcher network is a direct function of the number of CAS blocks.

CAS design consists of an n-bit comparator and two n-bit multiplexers (n is the width of input data). Figure 1.9 shows the conventional design of a CAS unit.

In the conventional binary design, increasing the data-width increases the complexity of the design.

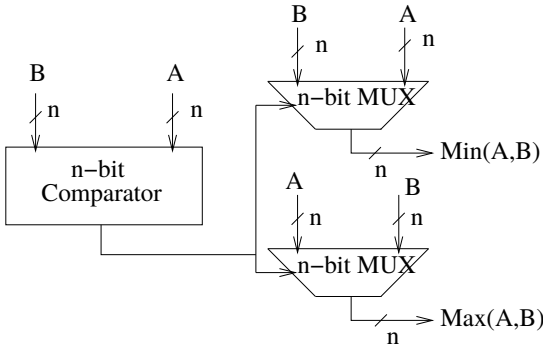


Figure 1.9: CAS block design

A bitonic sorting network sorts n elements in $\Theta(log^2 n)$ time.

We can easily build a sorting network to implement this bitonic merge algorithm.

Such a network is called a bitonic merging network.

The network contains $logn$ columns.

Each column contains n/2 comparators and performs one step of the bitonic merge.

Replacing of *min* and *max* comparators by *max* and *min* comparators results in decreasing sequence.

A sequence of length 2 is a bitonic sequence.

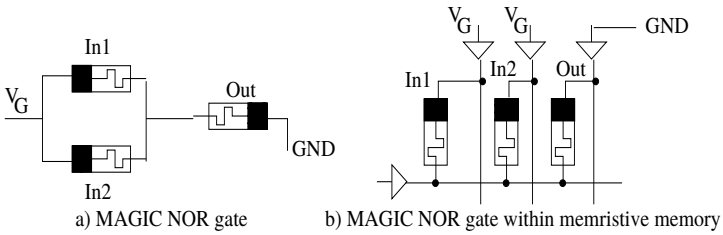
As the depth of the network is $\Theta(log^2 n)$

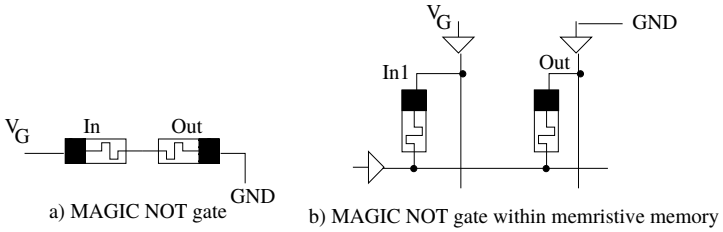
Each stage of the network contains n/2 comparators.

A serial implementation of the network would have complexity $\Theta(nlog^2 n)$.

Sorting in Memristive Memory

Use of MAGIC NOR operation.





NOR based 2-bit comparator is shown in Figure 1.10.

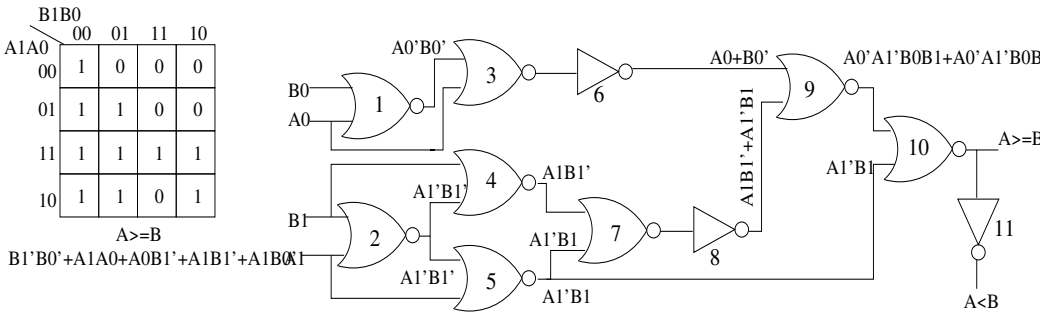


Figure 1.10: NOR based 2-bit comparator

A sorting unit (CAS unit) requires one comparator and two multiplexers.

Implementing an n-bit comparator by basic gates requires ($11n$ - 2) NOR and ($7n$ - 2) NOT gates.

Figure 1.11. shows in-memory implementation of sorting using MAGIC.

Each column contains n memristors (here, n = 2).

The computation includes NOR, NOT, and copy operations.

Each $G_{i,j}$ memristor in Figure 1.11 shows participation in a logical gate (operation) i in Figure 1.10.

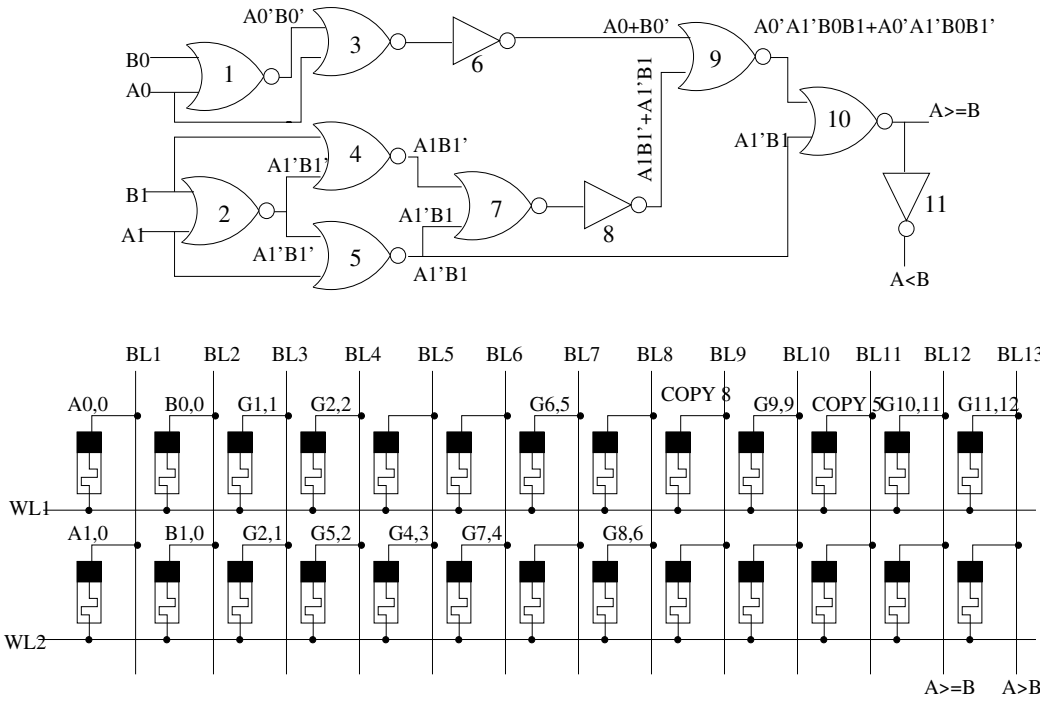


Figure 1.11: NOR based MAGIC design of 2-bit comparator

$C_{i,j}$s on the other hand, show the copy operation i, in which the state of G_i memristor is duplicated.

The index j marks the cycle number in which an operation is performed.

In some cases, a memristor participates in two operations.

For example, the memristor at right-bottom end.

To execute these operations, in each clock cycle, the memristor controller applies the proper voltage to crossbar columns and rows to execute some NOR or NOT operations concurrently.

All memristors with the same cycle number produce their output value at the same time.

After comparison, we need values of only four columns
(BL1 and BL2 for the two input data, and BL12 and BL13 for the two comparison results
to implement the multiplexer part of the sorting unit.

Hence, we could reuse the rest of the memristors.

NOR-based logic design of a multi-bit binary 2-to-1 multiplexer and in-memory two MAGIC-based 2-bit binary multiplexer for Max/Min selection is shown in Figure 1.12.

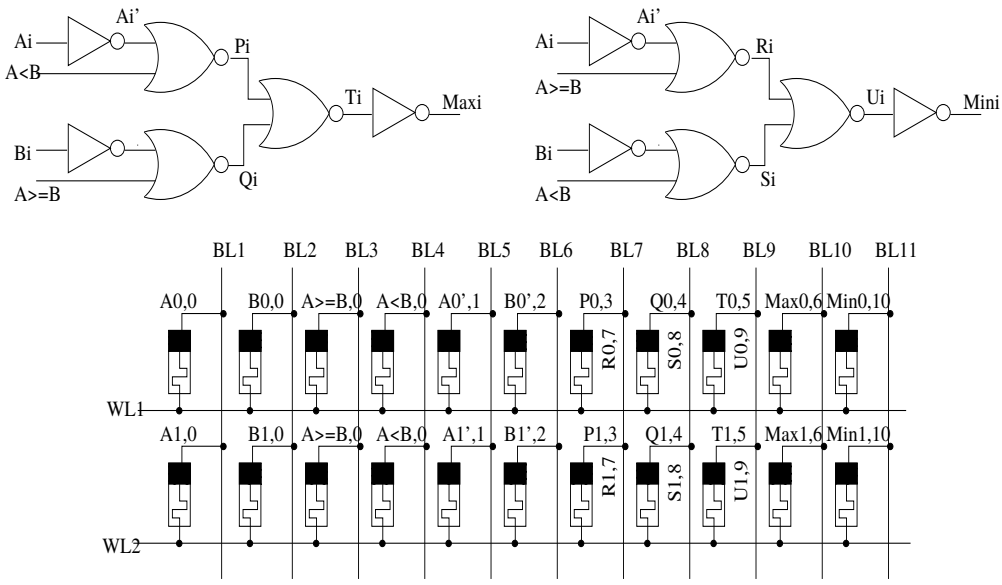


Figure 1.12: NOR based two MAGIC-based 2-bit binary multiplexer for Max/Min selection

In implementing the multiplexers, we re-use the memory cells of the comparison step.

The input data is inverted in two clock cycles, cycles 1 and 2 (on BL4 and BL5).

The first multiplexer produces the maximum value on BL10 in cycles 3 to 6.

The minimum value is produced on BL11 by the second multiplexer through cycles 7 to 10.

Since 3 columns used by first multiplexer (i.e., P, Q, T) are being re-used by the second multiplexer an additional cycle is considered for initialization before execution of next multiplex operation.

The execution of multiplexers, therefore, takes two initialization and 10 operation cycles.