# Sorting algorithm

## Theoretical background

Sorting algorithms are a fundamental part of digital processing systems. In hardware systems, specialized sorting logic can be used to preprocess or prioritize data in communication protocols, sensor networks, or real-time analytics engines. Sorting data streams on-the-fly allows for efficient classification, thresholding, or reduction of latency in decision-making systems.

The specific separation of positive and negative values, with placement on alternating indices, may be useful in signal processing pipelines or compression algorithms, where maintaining structured or grouped data is beneficial for subsequent processing stages.

## Goal

Your task is to create a module that will receive a data packet, store it, sort data in it, and send it back to the Judge in the following way:

- Positive numbers should be sent in the ascending order,
- Negative numbers should be sent in the descending order,
- If there are both positive and negative numbers to be sent, then positive numbers should be sent on even indexes of the output array and negative numbers on the odd indexes.

Table 1. shows an example data packet to be sorted. There're 5 positive numbers [3, 0, 2, 4, 1] and 3 negative numbers [-3, -1, -2].

*Table 1 Example of the input packet.*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | -3 | -1 | 0 | 2 | 4 | 1 | -2 |

Sorting [3, 0, 2, 4, 1] in the ascending order gives [0, 1, 2, 3, 4] and sorting [-3, -1, -2] in the descending order gives [-1, -2, -3]. Then the first 3 positive numbers are sent on even indexes 0, 2, 4 and 3 negative numbers are sent on odd indexes 1, 3, 5 and then remaining positive numbers are sent without index restriction, because there're no more negative numbers which could be sent on odd indexes. Table 2. shows an expected result for the input given at Table 1.

*Table 2 Example of the expected output.*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | -1 | 1 | -2 | 2 | -3 | 3 | 4 |

## Provided module

**You can edit only the *task_4.sv* file. If you wish, you may add other files for supportive submodules.**

You only get the outline of the ***task_4*** module. Inputs and outputs are declared as shown below in the Table 3.

*Table 3 List of inputs and outputs of the task_4 module.*

| Signal name | Signal type | Bit length |
|---|---|---|
| i_clk | Input | 1 bit |
| i_rst | Input | 1 bit |
| i_valid | Input | 1 bit |
| i_data | Input | 16-bit (signed, encoded as two's complement) |
| i_first | Input | 1 bit |
| i_last | Input | 1 bit |
| o_data | Output | 16-bit (signed, encoded as two's complement) |
| o_valid | Output | 1 bit |
| o_last | Output | 1 bit |

## Input and output interfaces

The task receives **a packet of 16-bit signed samples** on the *i_data* port. The **maximal amount of data is 4KB**. The beginning of the data block is marked by the *i_first* signal and the end by the *i_last* signal. Throughout the data transfer, the *i_valid* signal is high.

Task receives data in the way shown in the figure below:

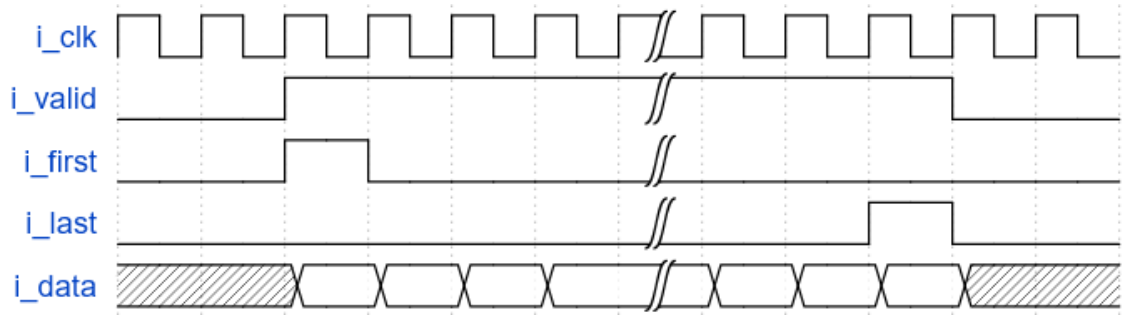


Figure 1 Input waveforms – the start of the packet (left) and the end of the packet (right).

The task should return a sequence of data using the *o_data* port. Along with the data the *o_valid* signal should be set, and on the last cycle of data the *o_last* signal should go high.

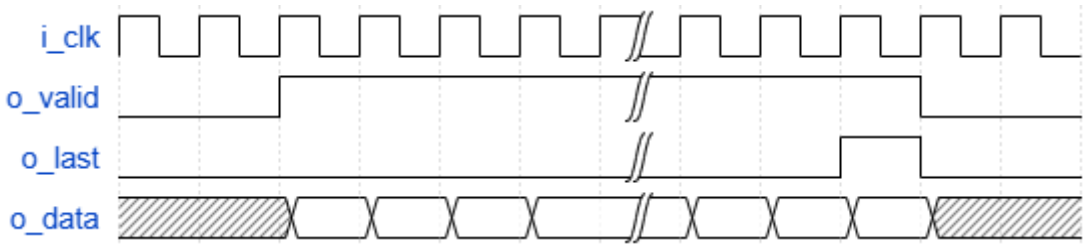


Figure 2 Output waveforms.

## Evaluating the task

The evaluation is done by comparing the sequence returned by the task with the expected sequence calculated by the reference model in the Judge. If the sequences are the same, the task is marked as correctly solved.

In Development mode, the task will be tested using a single set of data that is randomized on each run. In Evaluation mode, the task will be tested using 3 fixed sets of data, which remain the same across all runs and are identical for every team (for more information, check chapter 2.3.1 **Testing modes** in the **FPGA_Hackathon_2025_Project_Guide** document).

You earn 4 points for correctly processing each test vector. In Development mode, this value simply indicates that the task passed the test. Evaluation mode, your total base score is calculated by multiplying 4 by the number of test vectors, so you can earn up to 12 points.

Additional points may be awarded for **resource utilization and latency** (for more information, check chapter 2.3.2 **Bonus Points** in the **FPGA_Hackathon_2025_Project_Guide** document).