Design of IP core for Merge Sort FPGA acceleration

Fabrizio Maria Aymone*

Politecnico di Milano
fabriziomaria.aymone@mail.polimi.it

January 11, 2023

Abstract

FPGA have demonstrated on multiple occasions to offer the advantages of hardware acceleration of heavy workloads, without requiring the costly production of an ASIC or the power consumption of a GPU. The merge sort algorithm sorts a sequence by repeatedly performing simpler comparisons between sub-sequences, which at each iteration grow in size. Given the parallel nature of the algorihtm, it is possible to accelerate it creating a custom IP block to run on a FPGA. This report details the method adopted in designing the IP block and the results obtained when mapping it onto a PYNQ-Z2.

I. Introduction

FPGA represent one of the most promising alternatives for hardware acceleration of heavy computational workloads.The prerogative of such devices is their reconfigurability which allows the user to adapt the system to his specific needs. In comparison with GPUs, which leverage the parallel work of thousands of cores, FPGA consume far less energy which makes them suitable for low-power edge applications such as autmotive and telecommunications. On the other hand, ASICs are the optimal solution in terms of latency and energy as they target only a particular workload. However, their usage is severely restricted to the task they were originally designed for and their production requires a silicon fab resulting in high costs. In such scenario, FPGAs position themselves between the performant ASICs and the power-hungry GPUs. As a consequence, several algorithms can be easily accelerated at hardware level by programming an FPGA.

*Third-year Electronic Engineering Student at Politecnico di Milano

II. The algorithm

The Merge Sort algorithm sorts a sequence of numbers in three steps. In the first step the sequence is divided recursively in halves until obtaining a set of sequences composed by only one element, which can be considered by principle to be ordered. In the second step, each couple of ordered sequences is compared in order to produce an output ordered sequence whose length is the double of the original sequences. Lastly, the process is iterated with the new generated set of ordered sequences, until obtaining the final ordered sequence. The comparison operation performed on two ordered sequences can be massively parallelized, bringing a substantial speed up to the algorithm. By leveraging this observation, the following sections will illustrate the various IP blocks design adopted for acceleration of the Merge Sort algorithm.

III. Comparator tree

The first design that was formulated consists in a cascade tree of comparators. The input axi stream is fed to a SIPO bank of registers

whose value is stored, when T_LAST gets high, in a series of FIFOs. The FIFOs are read by the first line of comparator blocks who, on their turn, feed the second line of comparators, and so on. The output data comes from the FIFO contained in the last comparator block - when it is not empty, T_VALID is high, when it is sending in output the last element, T_LAST is high. This architecture is the fastest one, however it uses the most HW (2048 comparator blocks), exceeding the constraints of the PYNQ board for a sequence of 1024 elements. The comparator tree was successfully implemented on FPGA for a sequence of 256 elements.

IV. Comparator row

The previous design was not implementable on the PYNQ board as the implementation threw the error "DRC UTLZ-1", concerning resources over-utilization. Therefore, an alternative design was formulated in order to reduce the comparator blocks needed. The first part (SIPO and FIFOs) remains the same, but, instead of having the comparator tree, only the first row of comparators is used. The output of the comparators is then fed again to the same line and so on, until the first comparator block stores the final ordered sequence. This process of comparing and copying is orchestrated by a FSM. Even if this design requires half the comparator blocks used in the previous, i.e. 1024, it was still not enough to meet the resource constraints of the PYNQ board and the same error is thrown during implementation.

V. Hybrid (two comparator trees)

To further reduce the HW required, a hybrid approach was adopted. Two trees that process sequences of 32 elements are collocated in series. The first trees, processes the first 32 elemenets of the sequence and the output is stored in a FIFO. Then, the SIPO registers shifts by 32 elementes and the process is iterated. Eventually, there will be 32 different FIFOs containing sequences of 32 ordered elements. This FIFOs are fed to the second tree which outputs the final ordered sequence. All these processes are orchestrated using a FSM. This architecture requires only 128 merge blocks, however the implementation still throws an error concerning resource over-utilization.

VI. Discussion

None of the three different approaches yielded an implementable architecture on the PYNQ-Z2 restricted budget. The reason for that is still to be determined, however the number of comparator blocks is probably not the problem, as it has been already hugely reduced. An alternative approach consists in reducing the bit size, i.e. 32, to 8. Nevertheless, the DMA does not support memory mapped data widths below 32. As a consequence, AXI stream data width converters are needed. By using this method it was possible to realize the implementation and generate the bitstream. The sequence received, when using python notebook, showed strange results consisting of an ordered sequence of high numbers and zeros. The reasons for that should be further investigated.

VII. Conclusion

In conclusion, three different approaches were formulated and implemented. All of them, however, threw an over-utilization error during implementation for sequences of 1024 elements. Nonetheless, the comparator tree was successfully implemented for sequences of length of 256.

VIII. Github repository

At the following link it is possible to find the github repository containing the implementation described in the report. https://github.com/fabrizioaymone/merge-sort-fpga