Report

Darkhan Baizhan

201525145

Aim

The aim of the project was to build custom IP using Xilinx Vivado, which would sort the array of the size of up to 1024 in ascending order.

IP Architecture

The sorting was performed using Bubble Sort Algorithm, and two consecutive numbers were compared and if needed swapped.

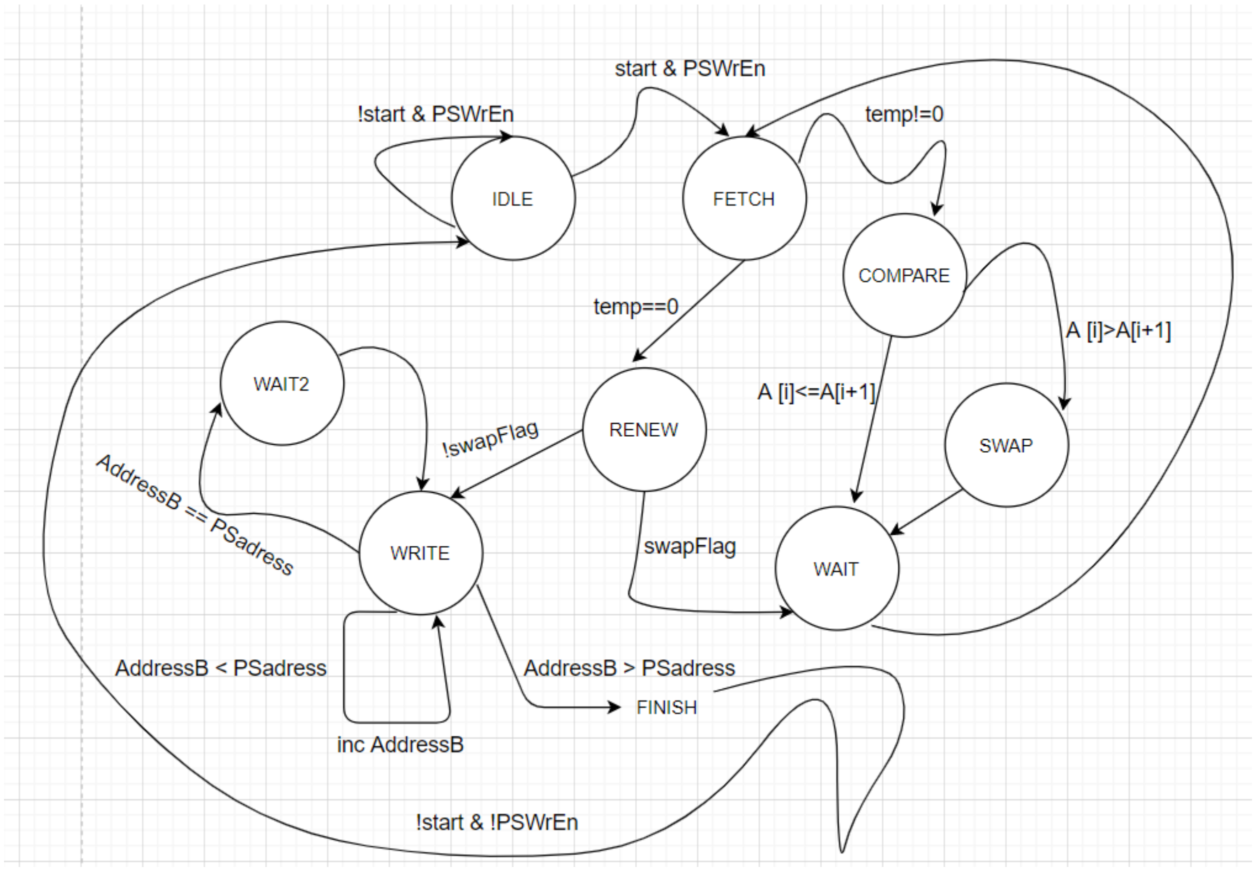
In the custom IP (mySortCore_v1_0) itself to IPs from Xilinx IP catalog were used (BRAM generator, FIFO generator)

BRAM was used for storing the array and then consecutively fetching the subsequent element and comparing them. It is worth to note that at the best-case scenario BRAM's reading clock latency is +1 clock. Therefore, in order to PS part to read from IP, one of the straightforward solutions is to make read latency 0. Consequently, FIFO with 0 read clock latency was used in the system.

FIFO had the purpose of storing the sorted array and when requested to export to the PS serially.

Algorithm:

FSM



The purpose of WAIT is to deal with one clock read latency and to wait one clock, while the algorithm proceeds to subsequent elements.

As it can be seen FETCH,COMPARE,SWAP WAIT represents the inner loop of bubble sort, while FETCH,RENEW,WAIT represents the outer loop. So, in a case if there was not any swaps during the whole inner loop (swapFlag == 0), then the array is sorted.

After sorting algorithm goes to 2nd part of writing to the FIFO through port B of the BRAM. Here is WAIT2 state is again introduced to deal with one clock read latency. PSaddress represents the last address where initially array was stored. So, the FIFO inouts would be until the PSaddress and in an case if Address of port B (AddressB) would become greater than PSaddress, it will mean that sorted array is in the FIFO, the done signal would become high and algorithm would be stuck at WRITE state, until start becomes low and PSWrEn == 0. The latter statement is only in order not to write the new data accidentally to BRAM through PS part.

Register Map

Address offset	Name	Read or Write type	Description
0x0	Start reg	R/W	LSB controls when the sorting will start (1 ->start of sort).
0x4	Status reg	R/W	Shows the whether the sorting is completed. (1->the sorting finished)
0x8	Array reg	W	The input to the IP is serially imported from PS through this register
0x12	Sorted reg	R	The output of sorted array is serially exported through this register

Software Drivers:

There is something to note regarding the drivers. Despite the fact that drivers were successfully changed in the /drivers folder of mySortCore_v1_0. After including the .xci files of BRAM and FIFO and eventual re-packaging of IP, any information about software drivers were deleted from the component.xml files. Therefore, having the drivers for software testing there are two versions of c source files for testing (regarding and disregarding the drivers)

Performance metrics:

The performance of the IP core is not perfect, for instance for the array of 4 numbers {3,2,1,4} the 29 clock cycles were needed to sort them. One of the main reasons is using the bubble sort which has time complexity of O(n^2), another reason is of using two redundant states in order to wait one clock cycle for read operations. The number of clock cycles can vary for the arrays of the same sizes, depending on the precedency of the numbers inside them.

Changing the array to {4,3,2,1} has expanded the time to finish the algorithm to 42 clock cycles. Because, greater number of operations were needed to sort out given array.

Conclusion:

To sum up, the aim of the project was achieved, the IP core which sorts the numbers and return to PS part is completed. However, it is worth of noting that too many resources were used to resolve the problem. For instance, there is definitely more elegant way to solve the issue of the +1-clock cycle read latency in BRAM, the algorithm could have been optimized to have fewer finite states. Nevertheless, the working IP core was created.