Akzhol Baktiyar 201593835

# Project Report

## Algorithm

True Dual Port RAM configuration BRAM generator and common clock for both ports were used. Moreover, in every port primitive output registers were unchecked to obtain read latency of one clock cycle and Write First configuration was used because my IP needs to read new values. Array is sorted using Bubble Sorting. It is is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. Unsorted array is saved in RAM and neighbors iteratively checked. If next address value is smaller than current address value numbers are swapped by enabling write enable of BRAM. My IP's inner iterator is sort_counter and initiallised with size of array. After it becomes zero it makes second loop by setting address 0. In addition, if in one inner loop swap didn't happen it will finish sorting. Outer loop iterator is called counter and it decrement each inner loop finishes. If counter equals to zero sorting finishes. Afterwards, sorted data is written to FIFO, which has no latency to directly read from it when read enable is high. Once FIFO is empty it goes to IDLE state.

Table 1. **Bubble Sorting** (bold numbers are compared)

| | Address 0 | Address 1 | Address 2 | Address 3 | Address 4 |
|---|---|---|---|---|---|
| 1. Input Data | 7 | 2 | 9 | 3 | 5 |
| Swap | **2** | **7** | 9 | 3 | 5 |
| | 2 | **7** | **9** | 3 | 5 |
| Swap | 2 | 7 | **3** | **9** | 5 |
| Swap | 2 | 7 | 3 | **5** | **9** |
| 2. | **2** | **7** | 3 | 5 | 9 |
| Swap | 2 | **3** | **7** | 5 | 9 |
| Swap | 2 | 3 | **5** | **7** | 9 |
| 3. | **2** | **3** | 5 | 7 | 9 |
| No Swap =Done | 2 | **3** | **5** | 7 | 9 |

## Architecture

Table 2. **Register Map**

| Offset Address | Register Name | R/W | Description |
|---|---|---|---|
| 0x0 | Control Reg | R/W | Bit 0 indicates start sort operation |
| 0x4 | Status Reg | R/W | Bit 0 indicates completion of sort operation |
| 0x8 | Sorted Array Reg | R | Address for reading sorted array |
| 0xC | Unsorted Array Reg | W | Address for writing the array |

Control Register set 1 by software and set 0 when sorting is completed (fifoDone==1).
Status Register set 1 be hardware when sorting is completed (fifoDone==1) and set 0 by software.
ReadEn of IP modules becomes high when processor read from BASE_ADDRESS+0x8
WriteEn of IP modules becomes high when processor write to BASE_ADDRESS+0xC

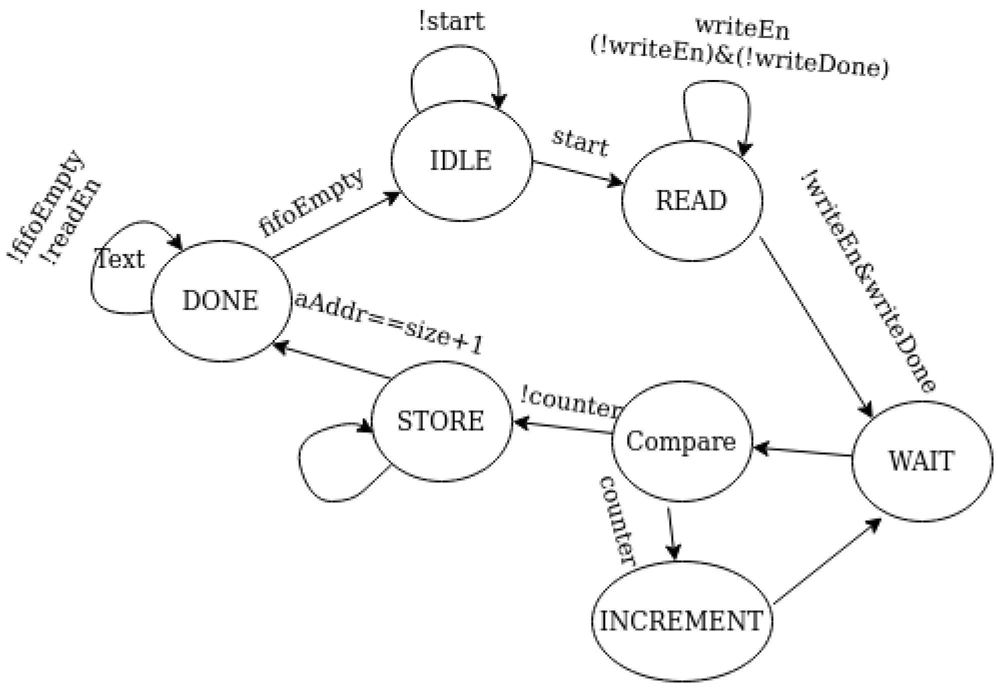
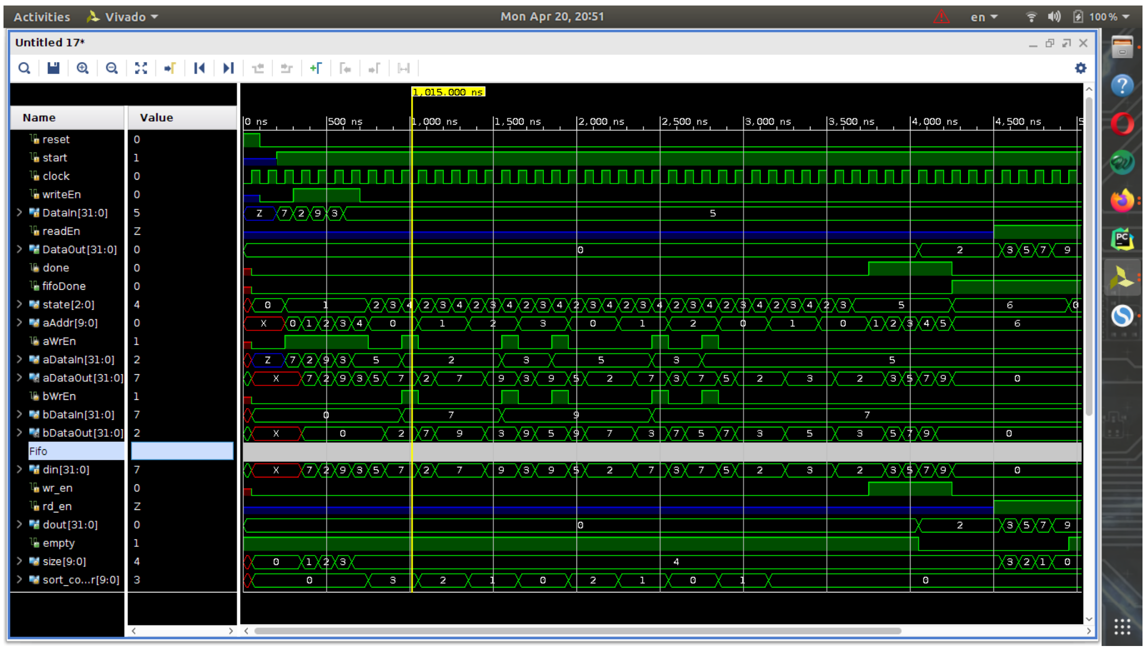


Figure 1. Finite State Machine

FSM States:

1. IDLE – idle state, does nothing until start is high
2. READ – reading of unsorted array coming from processor.
3. WAIT – to get correct numbers from addresses in COMPARE state
4. COMPARE – compares and swap numbers if needed, also checks if sorting is completed.
5. INCREMENT – iterates addresses, checks if inner is completed and is number swapped in last inner loop
6. STORE – to store sorted array in FIFO
7. DONE – indicates whether sorting is completed and sorted array can be read.



| Name | Value |
|---|---|
| reset | 0 |
| start | 1 |
| clock | 0 |
| writeEn | 0 |
| DataIn[31:0] | 5 |
| readEn | Z |
| DataOut[31:0] | 0 |
| done | 0 |
| fifoDone | 0 |
| state[2:0] | 4 |
| aAddr[9:0] | 0 |
| aWrEn | 1 |
| aDataIn[31:0] | 2 |
| aDataOut[31:0] | 7 |
| bWrEn | 1 |
| bDataIn[31:0] | 7 |
| bDataOut[31:0] | 2 |
| Fifo | |
| din[31:0] | 7 |
| wr_en | 0 |
| rd_en | Z |
| dout[31:0] | 0 |
| empty | 1 |
| size[9:0] | 4 |
| sort_co...r[9:0] | 3 |

Figure 2. Simulation

According to figure 2 first array is written to BRAM memory when write enable is high. Afterwards, sorting was made by swapping values in neighbor memories. Once sorting is Done it is stored in FIFO and when read enable is high fetched from FIFO.

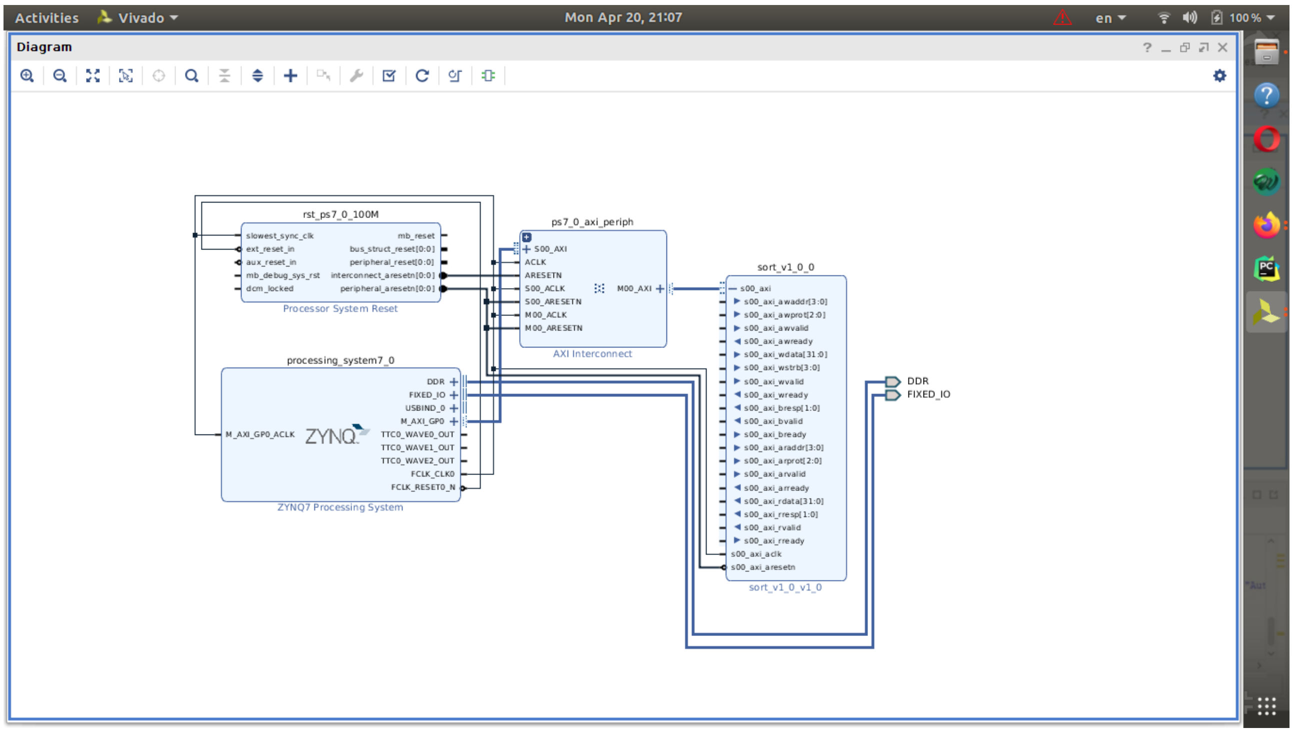


Figure 2. Block Design