Lab 5

Using an FPro SoC with Standard and Custom Hardware IP Cores: Fast Sorting

Design, implement and verify an FPro system and a corresponding application in C/C++ capable of performing sorting in software and hardware.

Your solution should support sorting of N w-bit unsigned integers for at least the following values of k, $N=2^k$, and w:

k	N	w
4	16	8
8	256	8
9	512	16
10	1024	16
11	2048	16
12	4096	16
13	8192	16

Switches SW3...SW0 should be used to enter $k=\log_2(N)$, i.e., $\log_2$ of the number of elements to sort.
Here are the suggested values of k, and the corresponding values of N and w, to be used for different phases of your project:

1. Debugging: k=4 (N=16 elements of the width w=8 bits)
2. Demo: k=4 and k=8 (N=16 and N=256 elements of the width w=8 bits)
3. Timing measurements: k=9..13 (from N=512 to N=8192 elements of the width w=16 bits).

The application should support the following functionality:

After power-up, the system should wait for the first press of BTNR or BTNL. All remaining buttons should be ignored.

Task 1: Initializing Memory

Each time BTNR is pressed when SW12=0, two arrays, sw_data[ ] and hw_data[ ], stored in the Processor's RAM, should be initialized with the same N pseudorandom values generated in software using a Pseudorandom Number Generator based on the C functions rand() and srand().

Required for teams and Bonus for students working individually:

Each time BTNL is pressed when SW12=0, two arrays, sw_data[ ] and hw_data[ ], stored in the Processor's RAM, should be initialized with the same N values generated in software using the equation:

$$sw\_data[addr] = HW\_data[addr] = N - 1 - addr.$$

Task 2: Display Mode

In the Display Mode, for $N=2^k$ with k=4..8, a user should be able to browse the contents of the arrays sw_data[ ] and hw_data[ ] using the following user interface:

SW15=0 : browsing sw_data[ ]
SW15=1 : browsing hw_data[ ].

The circuit should display

- Current Address (using Seven-Segment Displays 3 and 2)
- Value in sw_data[ ] (when SW15=0) or hw_data[ ] (when SW15=1) at the position given by the Current Address (using Seven-Segment Displays 1 and 0).

BTNU should increment the Current Address in a wrap-around fashion (e.g., for N=16, "0F" should be followed by "00").
BTND should decrement the Current Address in a wrap-around fashion (e.g., for N=16, "00" should be followed by "0F").

Default for students working individually:

For values of N > 256, browsing should be disabled, and the Seven-Segment Displays should show values of k and w, respectively, expressed in the decimal notation.

Required for teams and Bonus for students working individually:

For values of N > 256, the following functionality should be supported:

SW13=0 : Displaying current address using displays SSD3-SSD0
SW13=1 : Displaying data at the current address using displays SSD3-SSD0

SW12=0 : Standard behavior described above
SW12=1 (when SW13=0) : Ability to change the current address one digit at a time.
BTNL and BTNR can be used to choose a digit (in a wrap-around fashion). BTNU means increasing and BTND means decreasing currently selected hexadecimal digit of the address (in a wrap-around fashion). BTNC means changing the current address to the new value. An attempt to change the current address to the value beyond the range 0..N-1 should be ignored.

Task 3: Sorting

Pressing BTNC (when SW12=0) should initiate sorting. Sorting should be performed in software and hardware. Sorting in software should be performed on the array sw_data[ ] stored in the Processor memory. Sorting in hardware should involve transferring input data from the array hw_data[ ] to the Sorting core, performing sorting, and transferring results back to the array hw_data[ ]. The processed numbers should be treated as unsigned integers and should be sorted in ascending order.
The total number of clock cycles required for software and hardware sorting should be measured and stored.
During sorting, "----" should be displayed on the seven-segment displays.

After sorting is completed, the seven-segment displays should show the number of mismatches between the results of sorting in software and the results of sorting in hardware. Thus, 0000 will represent a perfect match. For N=16 (0x0010), 0010 will mean that none of the corresponding locations in each array matches.
After another press of BTNC, the system should come back to the Display Mode.

Task 4: Cycle Count Mode

After sorting is completed, the position of SW4 should have the following meaning:
SW4=0 : Display Mode
SW4=1 : Cycle Count Mode.

In the Cycle Count Mode, the total number of clock cycles used for sorting should be displayed on the seven-segment displays.

The position of the switch SW14 should have the following meaning:
SW14=0 : displaying the least significant 16 bits of the Cycle Counter
SW14=1 : displaying the most significant 16 bits of the Cycle Counter

The position of the switch SW15 should have the following meaning:
SW15=0 : number of clock cycles used for sorting in software
SW15=1 : number of clock cycles used for sorting in hardware.

Task 5: Debug Mode

During sorting and timing measurements, no information should be transmitted using UART (and thus displayed on the console).

During the remaining phases of the operation of your application, you can use functions of the uart driver, such as disp(), to help you with debugging the operation of your application and sorting core.

However, please note that the primary mode of debugging the sorting core is through the use of a VHDL testbench and functional simulation using the Vivado Simulator.

Deliverables:

1. Block diagram of the sorting_core including a wrapper (folder: bd)
2. VHDL code of the sorting_core (folder: src_rtl)
3. A testbench used to verify the correct behavior of the sorting_core (folder: src_tb)
4. C/C++ code of the main application, including all subroutines (folder: src_app)
5. C/C++ code of the driver for the sorting_core (folder: src_drv)
6. A video demonstration (folder: video)
7. Report (see the details below) (folder: report).

Note: **Make sure to create a separate directory for each deliverable mentioned above. Do not submit any other files of the Vivado project!**

Report File:

A. Names of team members and their respective responsibilities.
B. Memory map of the sorting_core.
C. List of fully completed tasks.
D. List of tasks attempted but not completed (please describe shortly what is missing).
E. List of any deviations from the original specification.
F. Tables and graphs showing for each value of N between 2^9 and 2^{13} (powers of 2 only)
    a. Execution time in software and execution time in hardware (in clock cycles, using logarithmic scale for graphs)
    b. Ratio of the execution time in software vs. execution time in hardware
    c. Resource utilization (#Slices, #LUTs, #FFs, #BRAMs).
G. Analysis of results.
H. Difficulties encountered and lessons learned.

Required for teams and Bonus for students working individually:
Measure execution time separately for two initialization types. Analyze the obtained results. Determine if the execution time depends on the initialization type.

Contest for the Fastest Implementation of Sorting

Bonus points will be awarded to students who perform sorting (correctly) using the smallest number of clock cycles in hardware and/or software.
Possible optimizations:

- Faster sorting algorithms in software
- Efficient C/C++ implementation
- Faster sorting algorithms in hardware
- Efficient VHDL implementation.

Important Dates

	Friday Section	Monday Section
Related Lab Sessions	Friday 3/24, 3/31, 4/7, 4/14 2023 8:40-11:20 AM	Monday 3/27, 4/3, 4/10, 4/17 2023 9:00-11:40 AM
Deliverables Due	**Thursday** **04/20, 11:59 PM**	**Thursday** **04/20/2023, 11:59 PM**
Demo and Q&A	**Friday, 04/21** **Monday, 04/24**	**Friday, 04/21** **Monday, 04/24**