Bubble Sort – Power & Area Estimation and Hardware Implementation

1 Week

Motivation and introduction

In this exercise, you will synthesize and implement the bubblesort application and then download it to FPGA board and see the results on the UART terminal or LCD. For visualizing, the output of BubbleSort and some additional information is printed to the URAT interface. You can use t_print() for directing output to LCD or u_print() to UART. Remember, you need to add respective libraries. Using these frameworks, the Bubble sort algorithm which will be implemented using the two CPUs to form two versions:
Version1: basis CPU (*browstd32.pdb*)
Version2: optimized CPU (*browstd32opt.pdb*) which supports new instructions
For every part, that starts like "a)", "b)" … you have to mail the answers and asked files to **sajjad.hussain@kit.edu** and use the topic "asipXX-Session6", with XX replaced by your group number.

Exercises

1. Preparing and Simulating Version 1

1.1. You have to create the software and the hardware sub directories under "*Application*" for browstd32.pdb CPU. First, create a new project directory inside your *ASIPMeisterProjects* directory for the new CPU and name it "*browstd32"*. You can use a copy from the *browstd32* CPU project from the last session, but do not forget to adjust the "*env_settings*". Remember, for LCD interface you need to create two separate subdirectories for the software and the hardware application in "Applications" directory.

1.2. Copy the provided and "*app_UART.c"* from "/home/asip00/Sessions/Session7" to the created LCD or UART subdirectories respectively. This file directs the printing to UART, you can change it to LCD by replacing u_print() to t_print(). However, UART interfacing is sufficient.

1.3. Copy the C libraries from "*asip00/epp/StdLib*" to each application subdirectory. For LCD simulation in dlxsim and ModelSim, use "lib_lcd_dlxim.c". For real LCD implementation in FPGA use "lib_lcd_320.c".

1.4. Also, copy the "*Makefile*" to both the subdirectories.

1.5. Make sure that you already have generated the VHDL files and GNU Tools for your CPU.

1.6. Compile ("*make sim*") the application for basis CPU and generates the required .dlxsim and DM/IM file for the dlxsim and ModelSim respectively.

1.7. Simulate the application with dlxsim using "*make dlxsim DLXSIM_PARAM="-da0 –pf1 –ufBubbleUART.out"*". It will start the dlx simulator to simulate the compiled file generated in the previous stage. Here, you have to pass some parameters to dlxsim such as the LCD/UART file to print the outputs.

1.8. You can restart the CPU by pressing the "*reset*" push button on the small mini board on the FPGA board, but REMEMBER, that your array in data memory is already sorted after the first run, so the second, third … run will be significantly faster than the first one.

1.9. The BubbleSort framework is measuring the number of cycles for the execution of the bubbleSort methods. This measurement is done by a counter on the FPGA Board or in dlxsim/ModelSim respectively. This measurement only measures the bubbleSort method, but not the overhead for e.g. printing the result.

2. Implementing the Project

2.1. Create your ISE project as discussed in the session 4. Synthesize, implement and generate the bitfiles.

2.2. Then from the application direcotory run "make fpga" and "make upload".

2.3. You can check the results on the UART. You can open HyperTerminal using "hterm &".

a) If your design works correctly, find out the design statistics (critical path, maximum frequency and area)
b) Compute the accurate time (in ms) required to sort the 20 numbers. Use the number of executed cycles (printed on the URAT interface) and the max. CPU frequency on the FPGA board, where the sorting is still correct).
c) Analyse the time and find the critical path (see Chapter 6.5 of the Laboratory Script)

3. Power Estimation

3.1. During ModelSim simulation also generate the VCD files for mentioned frequencies (first with 50MHz and then with Max. Frequency found in the last session).

3.2. Create Xilinx ISE project to estimate power with XPower.

a) Determine the total and dynamic power.
b) Compute the total execution time (ms). You can use execution as the # of cycles multiplied by the clock cycle in ModelSim.
c) Compute the energy required. Fill in all these results in the table below e.g. *PowerReport.xlx* or *PowerReport.ods*.
d) Does using any instruction minimize the required energy? A version uses an application, which needs less number of clock cycles than another Version; is it also power and/or energy-optimized version compared to Version2?
e) Repeat a-d, but instead of taking the default of 50 MHz, use the individual maximum CPU frequency on which a CPU can run (You can get it from ISE_Benchmark). This frequency has to be configured in tb_brownie32std.vhd (search for CLK_PERIOD; e.g. 10 ns half period = 20 ns period = 50 MHz). XPower will automatically load this frequency from the VCD file.

4. Preparing, Simulating, Implementing and Power Estimation for Version2

4.1. Repeat the above exercises for the optimized version a), b), c), d), e).

4.2. Sample PowerReport.xlx or PowerReport.ods

	Total Power [mW]	Dynamic Power [mW]	Execution Time [ms]	Energy [nJ]
Version1 50 MHz				
Version2 50 MHz				
Version1 Max. Freq: -----MHz				
Version2 Max. Freq: -----MHz				

Next Session: Adaptive Differential Pulse Code Modulation (ADPCM)

Readings: Recall relevant information from Laboratory Script, ASIPmeister Tutorial and User Manual