Bubble Sort - Hardware Implementation

Motivation and introduction

In this exercise, you will learn how to synthesize and implement your project and then download it on FPGA board and see the results on the UART terminal. You will test your BubbleSort implementations on the FPGA Board. For visualizing the output of BubbleSort when running on the FPGA Board the resulting array plus some additional information is printed to the URAT interface. As you don't have a compiler for the DLX-CPU yet therefore C-Code is to be translated to assembly, for this a new framework is provided. In this framework the BubbleSort implementation is removed out of the created assembly file. The assembly file framework exists in two versions for simulating with dlxsim and ModelSim ("*Bubblesort_LCD_forSW.s"*) and for running on the FPGA Board ("*Bubblesort_LCD_forHW.s"*). The main difference is the implementation of the "*t_print*" method (see Chapter 8.5 of the Laboratory Script): the dlxsim/ModelSim implementation is just printing all characters to the console or to a file whereas the implementation for the FPGA sends additional control signals and waits for the corresponding answers from the UART. Using these frameworks, the Bubble sort algorithm which will be implemented using the two CPUs to form two versions:
Version1: basis CPU (dlx_basis.pdb) with basis bubble sort algorithm
Version2: modified CPU (dlx_bgeu.pdb) which supports new instruction "*bgeu*"
It is important to note that "*bs_bgeu_opt????.s*" will not work on FPGA board as "*make fpga*" always assume four NOPS in assembly code, which is lost after aggressively optimizations. For every part, that starts like "a)", "b)" … you have to mail the answers and asked files with a CC to your group members to **asip00@ira.uka.de** and use the topic "asipXX-Session2", with XX replaced by your group number.

Readings for the next session: Chapters 7

Exercises

1) Creating the applications

1. You have to create the software and the hardware application sub directories for dlx_basis.pdb CPU. First, create a new project directory inside your *ASIPMeisterProjects* directory for the new CPU and name it "*dlx_basis"*. You can use a copy from the *dlx_basis* CPU project from the last session, but do not forget to adjust the "*env_settings*". Also create a two subdirectory for the software and the hardware application in "Applications" directory.
2. Copy the provided "*Bubblesort_LCD_forSW.s"* and "*Bubblesort_LCD_forHW.s"* from "/home/asip00/Session5" to the created software and the hardware application subdirectories respectively. Also copy the "*Makefile*" to both the subdirectories.
3. Add your *_bubbleSort*-method implementation from "*bs_basis.s*" (Session 3) to these frameworks "*Bubblesort_LCD_forSW.s"* and "*Bubblesort_LCD_forHW.s"* for software and hardware verification respectively. While adding bubblesort implementation to these frameworks you have to consider the following points:
   - The function calling scheme (parameter passing, register saving, stack preparation …) from the compiler was somewhat different to the scheme that was used in the old framework from Session 3 ("*bs_Framework.s*" and "*bs_Framework_pipelined.s*"). We have adapted the function calling scheme for the BubbleSort method and added it to the new frameworks.
   - Search for "*_bubbleSort*" in the framework files to find the method for BubbleSort.

- The framework already contains the function calling scheme and the "ADD YOUR IMPLEMENTATION HERE" comment.
- First, you should add your BubbleSort implementation to the framework ("*Bubblesort_LCD_forSW.s*") for dlxsim/ModelSim to test it. After it is running you can copy-and-paste your implementation to the framework for the hardware.
- Dlxsim will write all the "*t_print*" output (including the control signals; each character is written to a new line) to the console. But you can also forward the output to a file and look at it after the simulation has finished. Use the "*-pf{fileName}*" parameter (*pf* stands for putchar file) therefore.
- Similarly, ModelSim will write all the "*t_print*" output to "*ModelSim/lcd.out*" and "*ModelSim/uart.out*".
- 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.
- 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.

4. Compile ("*make sim*") the application assembly file "*Bubblesort_LCD_forSW.s*" for basis CPU and generates the required .dlxsim and DM/IM file for the dlxsim and ModelSim respectively.
5. Simulate the framework "*Bubblesort_LCD_forSW.s"* with dlxsim using "*make dlxsim*". 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 file to print the outputs. For that, you have to type "*make dlxsim DLXSIM_PARAM="-da0 -pfputc.out"*" where "*putc.out*" is the file than contains the printed output. Note: the pipeline is by default has the value 4. You have to check "*putc.out*" file in order to see the result of your code and check the sorted array.
6. Generate ASIPMeister hardware and software description files for dlx_basis.pdb CPU and then simulate "*Bubblesort_LCD_forSW.s*" with ModelSim, which will use DM/IM files already generated. Compare the resulted array from "*putc.out*" and "*lcd.out*", it should be same.

2) Implementing Project Version1

7. Every ASIP project has "*ISE_Framework*" directory, as they all instantiated from the default *TEMPLATE_PROJECT* in "*/home/asip00/ASIPMeisterProjects*". This "*ISE_Framework*" directory contains all the required IPs and VHDL files to build you FPGA project. Remember to adjust "*env_settigs*" accordingly. Create Xilinx ISE project using "*ISE_Framework*" as discussed in Chapter 6 of the Laboratory Script, and add required source files. Synthesize, implement and generate programming file for the FPGA.
8. Compile your assembly file "*Bubblesort_LCD_forHW.s*" using "*make sim*" and then "*make fpga*", it will compile your assembly file (in this case "*Bubblesort_LCD_forHW.s*") and generate the required DM/IM file and combine them with your bitstream which is generated from the Xilinx ISE tool (note that "*env_settings*" will tell the Makefile where the ISE project folder is located). Finally, a new bitstream file containing your hardware CPU along with corresponding IM/DM files of your application will be generated in the folder "*BUILD_FPGA*". This bitstream must be used to configure the FPGA.
9. Upload the existing bitstream to the FPGA prototype board using "*make upload*" (note: this command does not generate a new bitstream) then you can check the results on the UART. You can open HyperTerminal using "hterm &" and adjust its settings like: Baud rate=115200, Stop bit=1, Data bits=8, Parity=None, COM Port=ttyUSB0 (for example), Newline at=CR+LF,
   a) If your design works correctly, find out the design statistics (speed and area, see Chapter 6.4 of the Laboratory Script)
   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) HINT: When you run bubble sort second time by just pressing the reset button, then it will be significantly faster, as the array in the memory was already sorted from the first run! You have to upload the Bitstream again for a second test

c) Analyse the time and find the critical path (see Chapter 6.5 of the Laboratory Script)

3) Implementing Project Version2

10. Create the software and the hardware applications for "*dlx_bgeu.pdb*", simulate and implement this version on FPGA prototype board using the steps 1-9.
    a) If your design works correctly, find out the design statistics (speed and area).
    b) Compute the accurate time (in ms) required to sort the 20 numbers.
    c) Find the critical path.
    d) Compare the design statistics between the two versions.
    e) How does this affect the execution time (in cycles and ms)?