Lab assignment-1

Aim:

1. Integrate a GPIO module to Zynq processor to drive LEDs on Zedboard.
2. Write an application to display different values on LEDs and verify it to be working.
3. Now add an adder with the existing system.
4. Write a software application to add 2 numbers and display their sum (computed using the adder) on LEDs.
5. Develop an accelerator which accepts start address, num of words as inputs and reads corresponding number of data from BRAM, adds them and displays on LED.

Learning Objectives:

1. Familiarity with Vivado IP integrator tool suite.
2. Familiarity with Zedboard.
3. Integrate an IP with processor and control IO.
4. Familiarity with writing software application on generated hardware.
5. Familiarity with accessing memory using acelerators.

Steps:

1. Invoke Vivado IP integrator tool
2. Click on New project.
3. Give a path and name of the project. Click on Next.
4. Select RTL project and tick the “Do not specify sources at this time” box.
5. In the next pane, click on Boards tab. Select “Zedboard Zynq Evaluation....” option. Click on next and then Finish.
6. Click on “Open Block Diagram” in the window that appears.
7. Right click and select “Add IP”
8. Add “ZYNQ7 processing system”
9. Add “GPIO controller” (gpio_0). Right click and select “Customize Block”. In the board interface pane, select the leds_8bits for GPIO.
10. Add “Adder”
11. Since Adder doesn't have a memory map interface, we need to add a memory map interface to be able to interface to the processing system. We don't have any dedicated IP for the same, so we use GPIO module itself for it.
12. Add 2 more GPIO instances. (gpio_1 and gpio_2)
13. Right click on gpio_1 instance and select “Customize Block”. In the “IP interface pane”:
    1. Tick “Enable dual Channel”
    2. Tick “All outputs” for both the channels
    3. Set GPIO width to 15
14. Right click on gpio_2 instance and select “Customize Block”. In the “IP interface pane”:
    1. Tick “All inputs” for both the channels
    2. Set GPIO width to 16
15. After having placed all the blocks, now click on “Run block automation”. It will connect various IPs to the processing system and will include any additional block if needed.
16. Run “Validate Design” to check if all connections are proper.
17. You will see 2 errors for input ports not connected at adder.
18. We need to tie CE port to '1' to keep clock enabled. Add IP named “Constant”. Customize to make const width and const val as '1'.
19. We can connect the same clock as the processor output clock (FCLK_CLK0) to CLK pin of adder.
20. Connect the outputs of GPIO gpio_1 to A and B inputs of adder.
21. Connect the GPIO port at gpio_2 to output S of adder.

22. Again click on “Validate Design” and it should be successful. If not, fix the errors reported.
23. Save the block design by pressing Ctrl+S.
24. Now we need to follow the steps to generate bitstream from this block design. It requires:
    1. Synthesize Design
    2. Implement Design
    3. Generate bitstream
25. Once all the above steps are successful, the hardware bitstream is built and ready for use. Download the bitstream on the board by Open target and then download bitstream.
26. Now we need to write the software code which will run on the Zynq processor.
27. Firstly Click on File->Export->Export Hardware.
28. Click OK on the popup that appears (local to project).
29. Now click on File->Launch SDK
30. Once SDK window appears, click on File->New->Board Support package. Click on Ok/Finish without changing any settings.
31. Once BSP is created, click on File->New->Application project.
32. Give a project name of your choice (e.g. add_led). In the OS drop menu, select Standalone. In the Board Support Package, click “Use existing” and ensure that the same name as BSP created earlier is present.
33. In the template, you can select the Hello world example.
34. Now, you can write the application code. You can use driver functions from BSP to access them.
35. You can learn about debug using breakpoints and single stepping in this experiment.

Lab assignment-2

Aim:

1. Experience accelerator design in a system level scenario using matrix multiplication as an example.

Learning Objectives:

1. Familiarity with SDSoC tool.
2. Experience mapping a function dynamically on SW versus HW and design space exploration.
3. Learn various directives which can be used to control the structure of logic generated and understand the tradeoff involved in each.

Steps:

1. Open C://Xilinx/SDSoC/2014.4/docs/ug1028-sdsoc_getting_started.pdf
2. Go to chapter-5, Lab-1.
3. Perform the steps as given. Instead of using zc702 as the target platform, you are supposed to use zed as the target platform.
4. Now go to Lab-4 and do the exercise to understand the performance estimation flow.
5. Open the file mmult.cpp. You will see certain pragmas defined in the file. Explore the effect of removing some of the pragmas on the generated structure and the impact on latency. Few pragmas to explore are PIPELINE, ARRAY PARTITION. You can use the estimation mode to understand the effect of change of these parameters.
6. You can use the document ug1027-intro_to_sdsoc.pdf to understand details of some of the pragmas.
7. Now go to Lab-2 and do the exercise as explained in the lab. You should be able to get an idea about the impact on performance using different system ports.

Lab assignment-3

Aim:

1. Multithreaded execution of function, some threads on hardware and others on dual-core.

Learning Objectives:

1. How to write multi-threaded application for dual-core.
2. How to generate multiple hardware instances which execute in parallel.
3. How to execute different threads with some running on hardware and others on software.
4. Understand aspects like synchronization in threads.

Steps:

1. Start with a given application which calls 4 different functions, which can be executed in parallel.
2. Run the following cases and understand the difference in the total execution cycles:
    1. Run the 4 functions sequentially and note the total execution cycles.
    2. Run the 4 functions as threads on processor and note the total execution cycles.
    3. Run the 4 functions as threads – 2 threads spawned on dual processor and 2 others on hardware. Note the total execution cycles.
    4. Run the 4 functions as threads – 2 threads spawned on dual processor and 2 others on hardware (2 instances of HW). Note the total execution cycles.