Lab 1A Handout 2021

Starter Lab

Lab Overview

Lab 1A is the starter lab, designed to introduce the development tools used for creating hardware/software interfaces. After completing Lab 1A, you will have created an RTL project using the **Vivado** design suite from Xilinx. Your basic RTL project will consist of the soft-core **Microblaze**[2] processor running on the **Artix A7-100T development board**[1] from Digilent. The Artix A7 board utilizes a Xilinx FPGA chip, the Artix 7. The Artix A7 board was previously named Nexys 4 DDR, and is identical to the Nexys 4 DDR in terms of functionality.

The Microblaze IP core is configurable, and permits creation of a system with different memory sizes, clock speeds and # of pipeline stages. The goal of this lab is to use the Vivado design Suite to create a custom hardware design for a processor and its peripherals, export the hardware to the Xilinx SDK, and then develop **C** programs to run on the custom processor. The **C** programs will gather timing information about the embedded system in terms of the number of clock cycles to execute different operations. Finally, timing uncertainties in the gathered data will be statistically analyzed.

1 Equipment

Throughout the course we will be creating Vivado RT (Register Transfer) projects, which are based around the block level diagram. An RT project refers to the level of abstraction selected for describing the design.

Hardware Nexys A7-100T Digilent Development board with Artix-7 FPGA

Software Vivado 2019.1

Software Xilinx SDK (with Vivado Integration)

The **Lab Hardware Handout** provides information about setting up the Artix A7 Development board with Vivado 2019.1 and the Xilinx SDK. It also contains information on interfacing the DDR2 memory with Microblaze, and general information which is useful for all of the labs. For this lab, the DDR2 memory will need to be a part of your design. Read the **Lab Hardware Handout** before starting the exercises.

2 Microblaze Configuration to be Explored

The configuration of the Microblaze processor provides various options, such as the size of the BRAM memory used by Microblaze, adding data and instruction caches and enabling debugging tools. For this lab, build Microblaze using the configuration instructions in the Lab hardware handout. Follow instructions in the **Lab Hardware Handout** to start Vivado and create a project containing a Microblaze processor along with DDR2 Memory. For this lab you should run your program in the DDR memory with caches enabled.

3 Adding Peripherals

The Microblaze processor provides a central processing unit for our embedded system to interact with the physical world. The next step in the design of our system is to begin adding I/O peripherals. Follow instructions in the **Lab Hardware Handout** to add IP for: (1) Two AXI Timers and (2) LEDs. You should add the other peripherals in the hardware handout, but they are not used in this lab. Peripherals communicate with the processor using buses. In our design we are using the AXI bus. Buses are one of the design elements which face timing issues. As we complete the timing analysis of operations listed in Section 5, we begin to see when peripherals are competing for bus access.

4 Timing Analysis Guidelines

After building the hardware and generating bitstream in Vivado, export your hardware and launch SDK. Create a new "blank" Application Project in SDK. We have provided default source code files that you will need for this lab. Copy these files to the "src" directory of your Application Project. The main code for this project that will be edited by you is enclosed in timing.c. This code provides you with the basic structure for exercising different peripherals and for measuring the number of clock cycles.

Timing measurement on a computer is rather more complex than you might think. Abstractly, we can just read the timer, then do something and then read the new timer and voila... In real life, the memory access to the data and the instruction access are both non-zero time behaviors, the call to read the timer involves a context switch and stack update... In short, there is a lot of random overhead. In particular, timing a single operation or event is hard because you have so much random measurement overhead. One solution is to in-line write 5 events, 10 events, 20 events as so on – then do a regression to estimate the actual time it takes for events to run. A variable 'Data' has been setup in the default code to run such analysis on DDR read operations. Change the number of in-line calls in the code and fit a line (use linear regression) for 5, 15, 25, 40 in-line calls to estimate the measurement overhead and the per read timing of DDR memory. You can now use this estimation of average run time to offset your test measurements. This way you are estimating the actual run time of the function you want to test. Note the noise associated with this measurement. Do you see what you expect?

For the list below, measure each operation repeatedly using the built-in loop. Do not inline the timed operations, use the measurement overhead above to estimate the actual event times. Run the loop at least 10000 times to get a stab at some of the more rare event timings, for the USB port, run the loop 1000 times to keep the experiment time reasonable. Be careful not to simply increase the loop to much as the default data storage to the histogram can overwhelm the stack. If you want to run longer tests to catch rare events, you'll need to either merge multiple runs in the histogram or perform the histogram locally to prevent storing the measurements.

A histogram function that counts the number of samples in uniformly spaced, equally sized bins is included in the code. You may choose an appropriate value for total number of bins and use this function for statistically analyzing your data. Be sure to use enough bins to display the variance in the data.

Be careful about the number of variables you create on the stack – if the code suddenly stops running or halts, it is likely that you exceeded the available stack size, exceeding the data segment will cause loader or compiler errors.

5 List of Operations to be Timed

1. Timing of addition using integers
2. Timing of addition using floating point
3. Timing of writing the LEDs to turn on or off
4. Timing of reading a word from the DDR2 memory at a random location
5. Timing of writing to the USB Port: (**You need to run the loop for this part only 1000 times**)
   - (a) Write a floating point number using ***printf()*** (Always use printf() with a "\n", eg.: printf("A\n") to avoid BRAM overflow. fprintf also can work if "\n" is not working)
   - (b) Write a string of 10 characters using ***xil_printf()***

While you are gathering data, another method in main, extra_method() is also running. This extra_method() adds a source of resource contention, which will change the time for communicating with these peripherals.

6 Reporting Results and Observations

It should be taken into consideration that embedded systems are targeted at tightly constrained environments, with multi-kHz sampling rates, multiple interrupt sources, and relatively small memories with realistic bus environments. Write a two-page report, including the following results in your report:

1. Report the regression estimate of overhead and DDR timing from section 4.
2. Include histograms of the measurement data for the operations listed in Section 5.
3. Plot a CDF for the DDR access timing, What is the expected value for this timing? What is a 95% confidence interval for this measurement?
4. What do you observe about the magnitude of measured values and the timing uncertainties associated with different operations?
5. Did you have any outliers in your results? If so, what could have caused these values?

7 Video Submission Guidelines

Each group can work together on the lab assignment, but must submit the video demo individually. Upload a short video (do not exceed a minute) showing the following things:

1. What happens to the DONE pin (L021) before and after programming the FPGA.
2. The execution and output of timing.c for turning on LEDs (Sec 5.3)
3. The execution and output of timing.c for *xil_printf*() operation (Sec 5.5 (b))

Please identify the board user/videographer in the video by clearly stating your name clearly at the start of the video.

References

[1] *Artix A7-100T FPGA Board.* URL: https://store.digilentinc.com/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/.

[2] *ECE 153A - Course Microblaze Notes.* URL: http://bears.ece.ucsb.edu/class/ece253/MicroBlaze_Overview.pdf.