GCD: S1H — A Uni-Processor Hw/Sw Solution

(Laboratory Session 5, EDAN15)

Flavius.Gruian@cs.lth.se

February 28, 2017

1 Introduction

In this laboratory session you will have to integrate the hardware developed in the previous session into a larger system, similar to one you first built. Furthermore, you should write software containing the necessary communication and computations, to obtain a functioning hardware/software solution for the *gcd* of N-numbers problem. In addition you will also have to evaluate your design and report the device utilization, time (clock cycles), power and energy consumption, in a similar way you did for labs 2 and 3. The development and testing will be carried out using *Vivado 2016.x* and the *Digilent Nexys-4* board.

2 System Architecture

The architecture for this laboratory session is centred on a MicroBlaze cooperating with your *gcd* accelerator core. Starting from a typical uni-processor system (MicroBlaze, AXI peripheral bus, UART, and timer) you should connect your hardware into the system and get it running together with the processor.

Vivado supports custom IP cores, granted they are specified in a standard format. To be able to rapidly use your core in Vivado, an AXI-light slave template is available to you on the course web-page (`myip_v2.0.zip`). Unpack the archive inside your Vivado `project/ip_repo` directory. It should now contain a new directory called `myip_2.0`. Overwrite the `myip_2.0/hdl/user_logic.vhd` with your own `user_logic.vhd` (and copy any additional files, if you decided to use several modules and files) to add the functionality you implemented in the previous laboratory session. It is also a good idea to have a look at the other

VHDL files defining the IP, and make any adjustments in case you chose to have a different interface (for user_logic) than the one specified in the lab description. Furthermore, inspect the other directories present in `myip_2.0`, and especially the `drivers` directory, which provides a skeleton for the software drivers to access the core.

To make Vivado aware of the new IP, you need to rescan the user IP repositories. To do this, in Flow Navigator select **Project Manager→IP Catalog** to display the available IPs. Then right-click in the IP Catalog tab (right window) and select **Refresh All Repositories**. The new core should now be displayed under *User Repository / AXI Peripheral*. Add the new IP to your design and **Run Connection Automation** to connect it to the rest of the system. Also inspect the **Address Editor** to see the address range your core falls into. In reality you will never use the absolute addresses to access your core, but only the constants defined in the BSP, `xparameters.h` file associated with your processor in the SDK project.

All you have left to do now is resynthesize the hardware, export it to SDK, and then write the software that uses the IP mapped to memory addresses. To exchange data with the IP, the simplest way is to read and write to the addresses that map to the IP registers via C pointers. (Ask your lab assistant for guidance, if you cannot figure this out.)

3 Design Evaluation

You should use the same procedure for evaluating your system as you employed in labs 2 and 3. For a fair comparison, use the same data sets used in the aforementioned labs.

4 Assignment

To wrap it up, during this laboratory session you should:

1. Create the support architecture with your core in it. Make sure you set the core parameters right and that all signals are connected as they should.
2. Synthesize the whole design (generate bitstream and export to SDK), keeping an eye on the synthesis report to make sure it goes through without errors! Make a note of the device utilization data.
3. Ensure that the core works properly inside the system. Write the software part of the *gcd* application.
4. Carry out time measurements and record the number of cycles. Use the same data sets you used to evaluate your designs in labs 2 and 3.
5. Record the power and compute the energy consumption for the same data sets and compiler setups as used in the previous labs.
6. Include all the above data in the final report.

5 Final Remarks & Hints

- Verify with a few lines of code that the hardware receives/sends data before writing the whole program. You might use a *device ID* register in your IP, with a precoded value, that allows you to see the hardware is connected and can output values.
- You might need to reset the system (push the "CPU Reset" button on the board) before inputting data, to make sure your FSMs start from a correct state.
- You may use several instances of your IP, to increase the parallelism and speed up the computation. Naturally the software will need to monitor all of these then.