Prof. Dr.-Ing. Dr. h.c. J. Becker

Digital Hardware Design Laboratory (DHL)

Exercise 4- Interfacing of Modules to the ARM's Bus System

Institut für Technik der Informationsverarbeitung, Karlsruher Institut für Technologie (KIT)

1 Preparation

Prior to the laboratory afternoon for this exercise you should get familiar with the following topics:

- The complement on two representation for binary signed integer numbers
- How can the "School Mathematics" method for dividing integer numbers be applied on binary numbers?
  - Calculate the binary integer division with the values 42 / 7 using pen and paper
  - How would the steps for the calculation have to change if a negative value was used, e.g. a = -42, b = 7?
- What does "pipelining" mean in the field of hardware development?
- See also DHL slides on "Advanced VHDL and Pipelining"

2 Task Description

During the course of this lab, you will develop an integer divider IP-core just by knowing the requirements and test it using with a simulation. Secondly you will interface it to the ARM Processor and test it with the provided Software.

2.1 Exercise 1 – Implementation of a Signed Divider

In the first part of the exercise a signed divider needs to be implemented which has the following characteristics (see also the template "div16.vhd"):

- Use generics to parametrize the widths of the input- and output values
- Use the following widths as default values:

- Input1: 17 Bit std_logic_vector (to be interpreted a **signed**)
- Input2: 8 Bit std_logic_vector (to be interpreted as unsigned)
- Result: 9 Bit std_logic_vector (to be interpreted a **signed**)

Implement the signed divider using **"School Mathematics" for binary polynomial Divison** of Input1 divided by Input2. The divider should have a pipelined architecture and accept a new set of input values in each clock cycle.

Here, it is of special importance to recall **the two's complement representation** and decide on a strategy how to perform the division step by step and which information to remember during the computation in order to ease the calculation in the end.

How many zeros do you need to append to the divisor in order to apply the "School Mathematics" for binary polynomial divison step by step in Hardware? Using binary shifts of the divisor to iteratively determine the result of the division is the most efficient solution.

Simulate the divider with the provided **div16_tb Testbench** and verify that the results are correct. The first data set, the testbench passes to the divider consists of the values a=42 and b=7 which allows you to compare the divider results against the division you did by hand (see "Preparation" Section).

2.2 Exercise 2 – Packaging the IP as AXI-Bus peripheral

Package and test the IP to be able to use it as an IP in the Vivado Block Designs:

- *Tools → "Create and Package IP"*. Name it "axi_divider".
- Select AXI-Lite Interface ("Create a new AXI4 Peripheral"). The user logic should supply four Slave Registers, accessible from the AXI Bus
- Select "Finish"
- For changing the packaged IP you need to right-click it in the IP Catalog and select "Edit IP Packager". This will open a separate Vivado project for the packaged IP in a new window.
- Copy the div16.vhd file to the source file location of the packaged IP (ip_repo\axi_divider_1.0\hdl) so it becomes a *local source file* and add the file to the IP Packager project like a normal source file.
- Make sure, that all VHDL source files used in the IP are listed in the *"Package IP"* window under "*IP File Groups*" in "*Standard / VHDL Synthesis*" respectively "*Standard / VHDL Simulation*". If necessary, add missing files manually.
- The slave register description can be found in the automatically generated user_logic file which may be called similar to "axi_divider_v1_0_S00_AXI", meaning this core is connected to AXI slave port zero of the AXI interface of the core. Instantiate the div16 component in the user logic file and connect it to the Slave Registers as described below.
- Keep in mind that the result register slv_reg3 is read-only for the AXI bus! Only the divider should be able to write the register.
- Don't forget to re-package the IP in the *"Package IP"* window after changing or adding source files! "Review and Package" Tap → Re-package IP

Name	**Adress (Base + x)**	**Byte 3** **Bit 31 – 24**	**Byte 2** **Bit 23 – 16**	**Byte 1** **Bit 15 – 8**	**Byte 0** **Bit 7 – 0**

Control Register	0x00 = slv_reg0				Bit0: en Bit1: rst
Input a	0x04 = slv_reg1		Bit 16: a(16)	a(15 downto 0)	a(7 downto 0)
Input b	0x04 = slv_reg2				b(7 downto 0)
Result	0x08 = slv_reg3			Bit 8: Result(8)	result(7 downto 0)

- For the Testbench "axi_divider_tb" you need to add the axi_divider to the project as a bare IP core which can be generated:
    - Open the IP catalog, search for axi_divider_v1.0 and double-click it.
    - Press "ok" and select "generate".

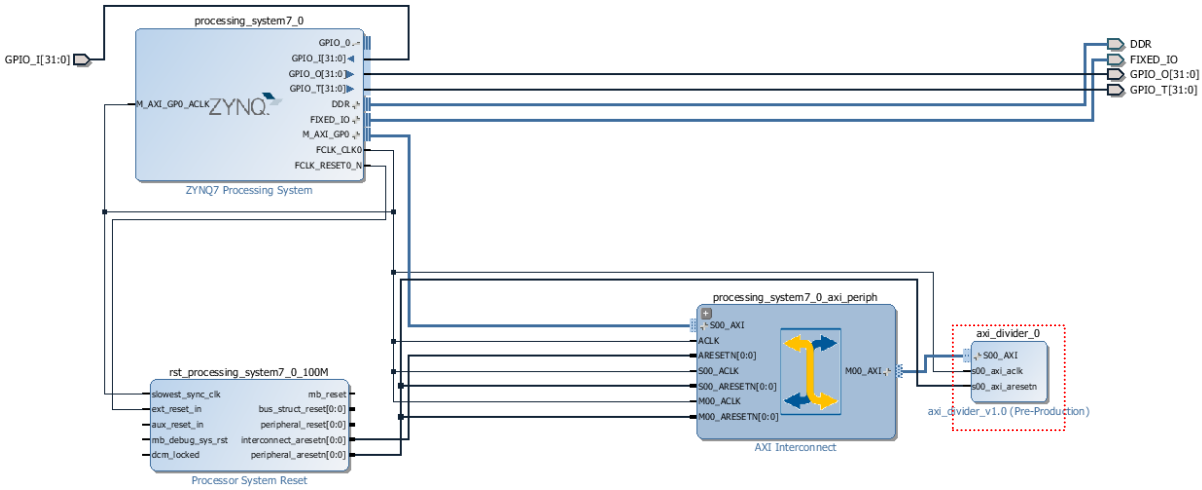
Simulate the packaged IP, now being called "axi_divider_0", with the Testbench being equipped with an AXI Traffic Generator, "**axi_divider_tb"**. The Testbench simulates an AXI-Bus an AXI-Master is connected to, where the AXI Traffic Generator performs the Master write accesses.

The AXI Traffic Generator is a core which can be interfaced with various AXI interface to initialize, test, etc. devices connected to the AXI Bus in simulation or in the running Hardware system. In the simplest mode of the AXI Traffic Generator (AXI Lite initialization), the core uses an address and a data file as input data set. A tuple of address and data defines an AXI access which is going to be performed on the bus after initial system reset. Like this, simple AXI-Testbenches can be implemented in order to test the operation of a packaged IP connected to the bus.

2.3 Exercise 3 – Hardware generation, Software compilation, testing on FPGA

After successfully simulating the packaged IP, add the IP to the Block Design, interface the IP with the AXI-Bus and generate a Bitstream file:

- Open the provided Block Design. It is equipped with an Zynq Processing System (ARM Processor) which is already configured for the ZEDBOARD with the desired System configuration, e.g. Clock frequency, Resets, Peripherals like UART, AXI-Bus Ports
- Add the IP to the Block Design "system_bd_i". IP Catalog → axi_divider →Customize IP
- Connect the AXI Bus of the Divider to the **M00_AXI Port** of the AXI Interconnect, the **FCLK_CLK0** of the ZYNQ7 Processing System and the **peripheral_aresetn** of the Processor System Reset IP.



- **Generate the Block Design**
- Open the address editor (*Window → Address Editor*) and auto assign an address to the divider (*right click on the axi_divider → Auto Assign Address*)
- **Generate the Bitstream** (first reset the Synthesis Results by *Bottom Window → Design Runs → synth_1 → Reset_Runs (Right Click)*)
- Export the Hardware files to the Software Development Kit (SDK):
  *File → **Export Hardware** (Include Bitstream)*

The Xilinx SDK is required in order to setup a Software Project, needed for compiling the source files in combination with the files comprising the definition of the properties of the components described in the block design like the Zynq peripherals and AXI devices and their address space (→ Hardware Platform Specification). In order to see the IPs and addresses known by the current Software project, have a look in the *.hdf file of the HW Platform Specification. The Board support package is the software framework which is built from the information specified in the HW Platform Specification, comprising all standard include files, the desired Xilinx libraries, and the files needed to use the standard Xilinx Zynq Peripherals or AXI IPs.

- **Launch SDK** with Workspace *"<Proj_dir>\SDK_WS"*
  - The simplest way to do so is to select "*File / Launch SDK*" in Vivado
- Create a new **HW Platform Specification**
  - *File → New → Project → "Hardware Platform Specification"*
    Call the Project "hw_platform_0", Select the .hdf file to be found in *"<Proj_dir>\VHDL_Ex3.sdk"*
  - Leave everything else as is
- Create a new **Board support package**
  - *File → New → Board support package*
  - Leave everything as is
- Create new **Application Project**
  - *File → New → Application Project*
  - Call the Project "Divider_prj"

- *Right click "Divider_prj" → Import →General → File System → Browse → "<Proj_dir>\VHDL_Ex3.data\sources\C_code"*
- Select "main.c", Advanced→ "Create Links in Workspace"
- Click Finish

- **Build the Project**: Right click on *"Divider_prj"→"Build Project"*
- **If needed,** Right click on *"Divider_prj"→ "Generate Linker Script" → "Generate"*

Now you will test the AXI Divider together with the compiled software on the ZEDBOARD.

- **Program the FPGA** with the Bitstream in SDK:
  Xilinx Tools → Program FPGA → Program
- In order to launch Software on the Zynq, the micro USB interface labeled with "UART" on the ZEDBOARD must be connected to the computer using a second micro USB cable
- **Open a Terminal program** (TeraTerm or the Terminal integrated in the Xilinx SDK), select the Serial USB Port of the Board. The baud-rate should be set to 115200 baud.
- **Send the Program to the FPGA and run** it:
  Right click on *"Divider_prj"→ "Run As" → "Launch on Hardware (GDB)"*
- In the terminal you should now see some outputs and a comparison of expected results to results determined by the hardware divider.
- If there are issues with the returned results on the console, **check the Simulation first**. *Debugging the logic in the running system takes a multiple of the time required for debugging the logic in the Simulation!*

Additional Notes:

- If you make a change to the Source files (here: VHDL files), the packaged IP needs to be **repackaged** in order to apply the changes to the packaged IP and the IP Catalog. ***Else, no change will happen*** if you modify the source files.
  IP Catalog → Right-Click the axi_divider → "Edit in IP packager" → Okay

  Check if the Source Files show the desired changes:
  IP File Groups → VHDL Synthesis

  In the end:
  Review and Package → Re-package IP

- For debugging Signals in the Running system add for each signal the following line to the VHDL Code
  "attribute mark_debug of Signal-to-be-debugged : signal is "true";
- An "*Integrated Logic Analyzer Core*" can be added to the project in the "*Synthesized Design*" view by clicking on "*Setup Debug*", adding the desired Signals to the Core (don't forget to save the changes!) and rerunning the Implementation/Bitstream Generation

- In Vivado you need to open the Hardware Manager

- Set the desired trigger conditions, arm the triggers
- Analyze the operation of the IP