Lab Exercise 4
System on chip
Implementation of a system on chip system on the Zynq

INF3430/INF4431 Autumn 2016

Version 1.2/06.09.2016

This lab exercise consists of 4 parts, where part 4 is compulsory only for INF4431 students:

1. Implement a simple application in the processing system (PS) and connect it to the leds and buttons on the physical board.
2. Implement an axi4 handler in programmable logic (PL), and connect this to the solution from the previous exercise to enable reading and writing to the DDR ram connected to the PS.
3. Integrate the solutions in PS and PL by allowing the user to read and write setpoints for the motor from the terminal. These are written to the shared memory and can be accessed by the PL.
4. INF4431: Use the Integrated Logic Analyzer to visualize the Axi communication between the modules.

Part 1 – Implement the processing system

Goal: The goal of this exercise is to successfully implement a bare metal application running on the processing system on the Zynq, which interfaces to the switches and leds on the ZedBoard. The leds should display a single lit led and BTNL and BTNR should move the location of the light to the neighboring leds on the board, including rotation. See Figure 1 *below.*

Required for approved delivery: The Vivado project with the implemented system, and a successful presentation of the working hardware to a lab supervisor.

At power up:

When pressing BTNL (left):

When pressing BTNR (right):

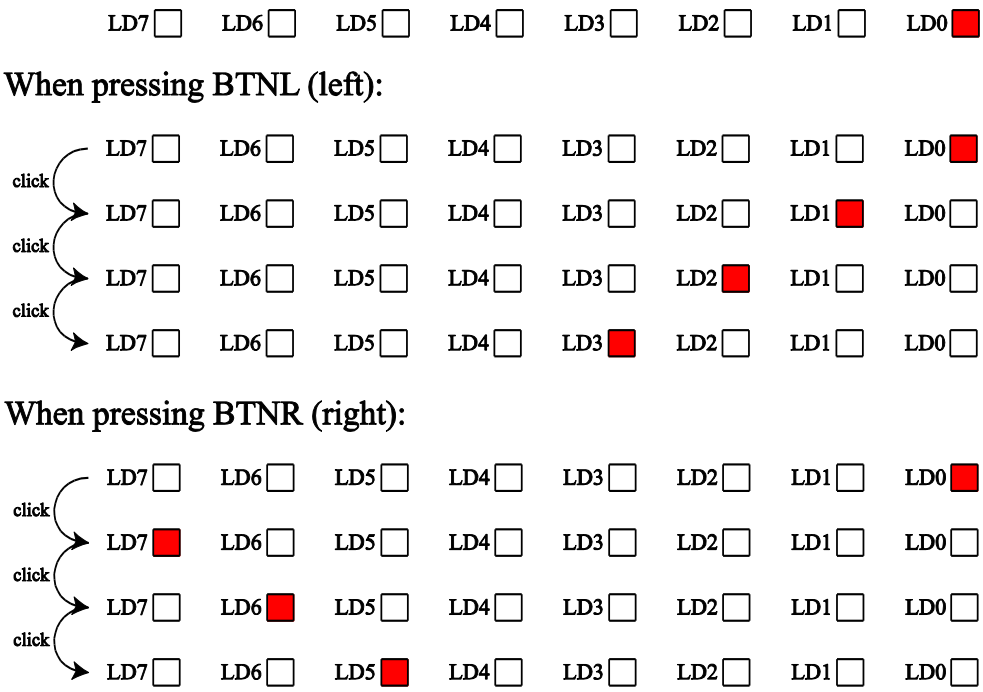

Figure 1 - Correct behavior for part1

Project and block diagram creation

- [ ] Create a new blank Vivado project called "Lab4_part1"

  The target language should be set to VHDL, and the ZedBoard should be selected among the boards during part selection, to ensure correct project settings.

- [ ] Run the following command to program the solution to the board:
  zynq_flash -f lab4_part1_solution.mcs -flash_type qspi_single

  Ensure that all MIO jumpers are set to GND before starting, and that the flash file is in the current Vivado directory. After programming, turn off the board, move the MIO5 jumper to 3v3, and turn it on. Use the solution to understand what the goals of the exercise ask for, and how to implement them.

- [ ] Create a new Block Design called "Lab4_BD" and insert a Zynq7 Processing System. Run Block Automation on the Zynq part.

  The block automation initiates the Zynq PS with correct settings for the chosen board or chip, as long as "apply board preset" has been chosen. Cross triggering is not needed for this lab, and can be kept disabled.

- [ ] Configure the PS to only use the UART1 peripheral, and a single general purpose AXI master port.

  Only these ports should be left on the PS:
  - DDR
  - FIXED_IO
  - M_AXI_GP0

- FCLK_CLK0
- FCLK_RESET0_N
- M_AXI_GP0_ACLK

☐ Add an AXI GPIO IP to the design and connect it to the PS

Since we will be interfacing to both buttons and leds, instantiate a dual channel block with leds on channel 1 and buttons on channel 2. Connection and constrains are handled by the Connection Automation. Ensure that the previously selected AXI master port is used through a correctly configured AXI interconnect.

☐ Generate a HDL wrapper for the block design and run synthesis, implementation and bitstream generation.

The HDL wrapper can be generated from the sources window, and adds the block diagram to the rest of the project. Right click the block diagram and choose "generate hdl wrapper".

☐ Export the hardware platform for the software development tools, and launch the SDK from the Vivado GUI.

Remember to include bitstream in the export operation, to allow programming of the board from the SDK. Launching the SDK from the Vivado GUI ensures correct workspace initialization and resource import.

☐ Create a new application project named "Lab4_part1", based on the "Hello world" application template.

The board support package includes constants and support functions used in the design of the application. Ensure that "standalone" is selected in the OS Platform box, as we are developing a bare metal application. The hardware platform exported from Vivado should be selected as Target Hardware, and the processor should be the first core of the onboard Cortex A9. The target language of the exercise files are C, so it is recommended to select this.

☐ Delete helloworld.c, and add the main.c from the lab exercise. Build the project using the "build all"-command in the Project menu.

☐ Make a new run configuration based on GDB that resets and programs the entire system, and name it "Lab4_part1". The board should now report the state of the buttons over the serial port.

Both ps7_init and ps7_post_config should be run to ensure proper initialization of the system. It is easier to debug when STDIO is attached to the console, but you can also open the connection in the terminal tab directly. The default baud rate is 115200, 8 data bits and 1 stop bit, no parity or flow control (though this can be changed in the BSP).

☐ Change the code in main.c to reflect the goals of the assignment: Only one led should light up, BTNL moves the light to the left, and BTNR moves the light to the right. The light should restart on the opposite side if moved off the side.

Port direction needs to be set, along with channel for the leds. The function XGpio_DiscreteWrite is used to write to the AXI GPIO module. Pressing F3 while selecting a function name takes you to the declaration and documentation.

Part 2 – PL and custom IP generation

Goal: The goal of this exercise is to successfully implement the programmable logic part of the assignment, based on the solution from exercise 5 in lab 3. The system should now be extended to include an AXI4 port to allow storage of motor setpoints in the RAM on the PS side. A test bench should be made to verify the solution and help debug during development, and the finished system should be tested in hardware.

BTNU should be used to save values, while BTND should be used to load values. The leds on the board should show the current memory slot used, 1-7. The save button should read the value from the switches, and write it over the AXI interface to the memory address defined by the current slot. An offset of 0x10000000 should be added to the memory addresses to save in a valid memory area. The slot is then incremented and the process can be repeated to save up to 8 set points to memory. The load button is used to load from the current memory slot, before incrementing the memory slot. This enables the user to play the eight saved set points from memory. See Figure 2 *below for a diagram of buttons, leds and switches, and the section below for loading the solution onto the board.*

Required for approved delivery: The Vivado project with the implemented system, a complete test bench with .do files, and a successful presentation of the working hardware to a lab supervisor.

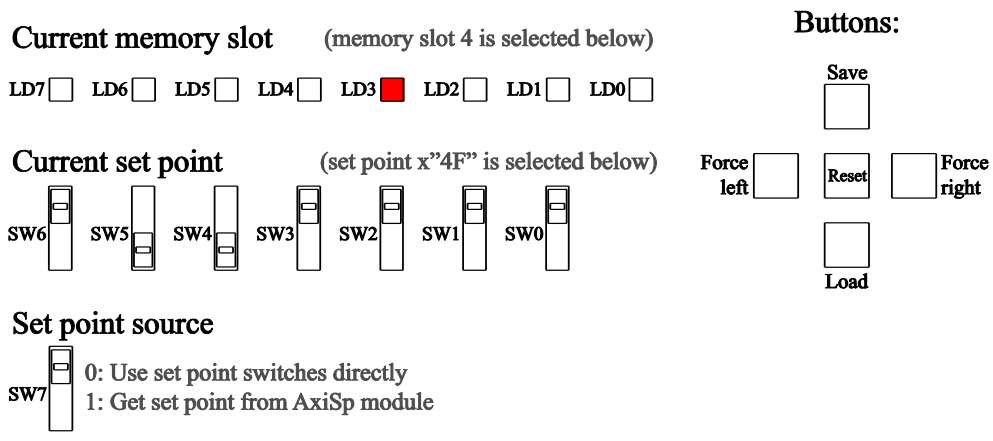

Figure 2 - Correct behavior for part2

Solution (optional)

- ☐ Create a new empty Vivado project called "lab4_part2_PL.

- [ ] Run the following command to program the solution to the board:
zynq_flash -f lab4_part2_solution.mcs -flash_type qspi_single

  Ensure than all MIO jumpers are set to GND before starting. After programming, turn off the board, move the MIO5 jumper to 3v3, and turn it on. Use the solution to understand what the goals of the exercise ask for, and how to implement them.

Programmable logic

- [ ] Create a new folder for the assignment called lab4_part2, and copy all files from the lab3 exercise 5 solution into a src subfolder.

  Include all .do, constraint or wave files.

- [ ] Copy AxiSp, debouncer, and Axi4LiteBfm files into the folder from the assignment zip.

  AxiSp is a template containing an Axi4 Lite interface, while the Axi4LiteBfm is used in the testbench to verify correct communication. The debouncer is used to ensure clean input from the buttons on the board.

- [ ] Open QuestaSim. Create a .do file to handle file compilation, testbench execution, and wave setup.

  Each simulation typically includes a sim.do file with compilation and execution, and a wave.do file for setup of the wave window. The latter can be saved by clicking the wave window and pressing ctrl+s.

- [ ] Add the axi4-lite, buttons and led interfaces found in AxiSp to the top entity.

- [ ] Instantiate the AxiSp module at top level, and connect it to the new ports created above. The set point for the rest of the system will come from the physical switches if SW7 is low, and from the AxiSp module if the switch is high.

  The selection between set point sources should be done through combinational logic alone, and will result in a single mux in hardware.

- [ ] Instantiate a debouncer module for each button input, and connect the outputs to the input of the AxiSp module. Define a generic integer in the top module that defaults to a value of 25.

  This allows the value of the debouncer to be overwritten during simulation, where the system is typically simulated for far less time than the real system.

- [ ] Set up the testbench with a new instantiation of the unit under test, and add new test cases for the functionality required in this exercise.

  The save button should be pressed 8 times with different setpoints on the physical switches, then SW7 should be set high and the load button should be pressed 8

times to load all the previously saved values. A debouncer counter size of 3 is sufficient for testing purposes, and should be defined from the test bench.

- [ ] Implement the AxiSp module rtl code so that it satisfies the requirements in the goals, defined above.
- [ ] Verify that the code works as intended in the test bench, using the Axi4LiteBfm.

IP creation

- [ ] Create a new Vivado project called "lab4_part2_PL", and include all files in the system.

  Do not include any constraints files. HDL files used only for simulation needs to be tagged correctly in the source file properties, available from the hierarchy tab in the sources.

- [ ] Move the top entity declaration inside the same file as the architecture, and remove the postfix from the file name.

  This has to be done since the IP packager of Vivado only supports analysis of a single top level file.

- [ ] Select "Create and Package IP" from the tools menu and complete the wizard.

  Package the current project, and choose lab4_part2_ip as the IP location

- [ ] In the Ports and Interface tab, correctly auto infer clock and resets, and infer the AXI interface.

  A sync_rst port automatically appears if no other reset is detected. Remove this if present. Right click the clock signal and auto infer single bit clock. Do the same for the two reset signals, but select single bit reset. Edit the two resets and make sure the correct polarity is given in the parameters tab. Add a new AXI master interface and pair up the correct signals in the port mapping tab. Make sure the axi interface is associated with your main clock through right clicking and selecting "associate clocks".

- [ ] Fill in other info where needed, choose "Package IP", and close the project.

Processing system

- [ ] Create a new Vivado project called "lab4_part2_PS", and close any open projects.
- [ ] Create a block diagram, and insert a "Zynq7 Processing System". Run through the designer assistance with the board preset.
- [ ] Remove all peripherals other than the UART1 and GP0 interfaces.

- [ ] Insert the “Processor System Reset” IP, “AXI interconnect”, and your custom IP.

  Your custom IP can be found by adding the IP directory using the IP Catalog.

- [ ] Connect the design together, and define external ports as needed.

  Use the clock output from the processing system, and not an external clock. If your design uses both synchronous and asynchronous reset, connect them together to a single external output. All ports that are external can be made so by right clicking them and selecting “Make external”.

- [ ] Mark the GP0 interface for debug.

  This is done by right clicking and selecting “Mark debug”, which adds small green icons to the cable.

- [ ] Add the constraint file from the src directory, and add LEDs and other missing constraints.

  The save button is set to BTNU, while BTND is the load button. BTNC should be connected to the reset port. Left and right buttons force the motor movement as previously.

- [ ] Generate a design wrapper, and choose “Run Synthesis”.
- [ ] Set up debug for the complete AXI interface with a depth of 8192 and a single pipeline stage.

  Ensure that every signal is associated with the clock coming from the processing system.

- [ ] Select “Generate Bitstream”. Export to hardware when done, and launch SDK.
- [ ] Create a new application project in SDK called “lab4_part2”, and choose the “hello world”-template.
- [ ] Make a new run configuration based on GDB that resets and programs the entire system, and name it “Lab4_part2”.

  Both ps7_init and ps7_post_config should be run to ensure proper initialization of the system. It is easier to debug when STDIO is attached to the console, but you can also open the connection in the terminal tab directly. The default baud rate is 11520, 8 data bits and 1 stop bit, no parity or flow control (though this can be changed in the BSP).

- [ ] Verify that the system works as expected, or go back to Vivado and QuestaSim to correct any errors.

Part 3 – Interfacing PL and PS

Goal: The goal of this exercise is to successfully implement a shared memory communication between the processing system and the programmable logic. The processing system should be able to read and write all set points in memory, while the programmable logic part works as it did in the last part. A boot image should be generated and programmed into the onboard flash memory to enable program execution after cycling of board power.

The program should present the user with two options. 'r' for read and 'w' for write. If the user runs the read command, a list of all 8 memory slots should be returned. If the user runs the write command, a memory slot should be requested, and data to fill the slot. A single digit data is sufficient (0-9).

Required for approved delivery: The Vivado project with the implemented system, and a successful presentation of the working hardware to a lab supervisor.

- [ ] Open up the "lab4_part2_PS" project, and save it as "lab4_part3" in a new directory.

  Make sure all files are imported so changes are done local to the new project.

- [ ] Run the following command to program the solution to the board:
  zynq_flash -f lab4_part3_solution.mcs -flash_type qspi_single

  Ensure that all MIO jumpers are set to GND before starting. After programming, turn off the board, move the MIO5 jumper to 3v3, and turn it on. Use the solution to understand what the goals of the exercise ask for, and how to implement them. To fully test the solution, connect a terminal to the board with a baud rate of 115200.

- [ ] Generate bitstream, export hardware, and open SDK.

- [ ] Open the helloworld.c application from the last exercise.

- [ ] Implement the program described in the goals above.

  The function "XUartPs_RecvByte" is used to receive a byte over the UART, and requires the base address found in the xparameters.h file in the board support package. Functions "Xil_In32" and "Xil_Out32" can be used to read and write to the memory. Documentation on functions can be found by pressing F3 while the function name is selected. Caching needs to be configured correctly, or disabled all together.

- [ ] Verify that the design works as described above.

  Suggested test progression: Read in terminal, write in terminal, read in terminal, read from board, write from board, then read from terminal.

Generate a boot image and program flash

- [ ] Open the SDK project used above.
- [ ] Right-click the lab4_part3 project in the project explorer and choose *Create Boot Image*.
- [ ] Add the fsbl.elf bootloader file from the assignment zip.

  Make sure the fsbl is at the top of the list, and of partition type 'bootloader'.
- [ ] Change the output path file name to 'image.mcs' and create the image.

  BOOT.bin is used for booting over SD card, while an mcs file is used for programming the on board flash memory.
- [ ] Choose *Program Flash* from the *Xilinx Tools* menu, and add the newly created image.

  Select qspi_single, and check boxes for blank checking and verification.
- [ ] Turn on the board, and press the program button.
- [ ] When the operation finishes, turn off the board, move the mio5 jumper to 3v3, and turn on the board. The *done* led should be lit after a second, and the program is running.

Part 4 – Using the Integrated Logic Analyzer (ILA)

Goal: Using the ILA to visualize the communication signals between the PS and the PL. Using and understanding Xilinx documentation to solve the task.

Required for approval: An exported image showing all AXI signals during transfer of a set point. Demonstration of the ILA setup for a lab supervisor.

- [ ] Review the theory from the slides on ILA or the videos from https://www.xilinx.com/training/vivado/ . Watching "Programming and Debugging Design in Hardware" is recommended at a minimum.
- [ ] Mark which signals to debug. This can be done in code, or in the GUI of Vivado. Use either the specific AXI debugging mechanic, or simply mark all AXI signals for debugging separately.

  Use Xilinx documentation including the videos and documents UG936 and PG172 to achieve this.

- [ ] Set the depth of the ILA to an appropriate level. Use your test bench to see how many clock cycles are needed if you are unsure.
- [ ] Re-synthesize and PAR with the new ILA core.
- [ ] Open the hardware manager and connect to the debug core.
- [ ] Setup proper triggers and start the core

  Triggers are the condition that starts the capture. Look at your test bench to see at what point your communication starts if you are unsure.

- [ ] Format the resulting waveforms and verify that the circuit behaves like in the simulation.
- [ ] *Optional:* Introduce an intentional error into the design and repeat the process. Verify that the system now does not work.