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

Digital Hardware Design Laboratory (DHL)

Exercise 2- Creating Testbenches for module verification

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

1 Preparation

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

- Fundamental knowledge on processors and machine code processing
  - See also additional material “Basics processor”
- Basics of HDL simulations
  - What is a testbench and for what is it used?
  - What possibilities do VHDL simulation tools (especially Vivado and ModelSim) offer in order to test and debug VHDL designs?
  - See also additional material “Testbenches and logic simulation in VHDL”

2 Task Description

2.1 Main Task

Write a Testbench for testing the ALU core provided as part of the templates. The Testbench should test the ALU using the Opcode Mapping [(7 downto 4) of 8-Bit Instruction word] depicted below.

Table 1: Operations of the ALU

#	Opcode	Operation	Explanation	Flags	Comments
0	0000	+	Addition	Z,N,C	data_in0 + data_in1 (not VZ-afflicted)
1	0001	-	Subtraction	Z,N,C	data_in0 - data_in1 (not VZ-afflicted)
2	0010	OR	log. OR	Z,-,-	

3	0011	AND	log. AND	Z,-,-	
4	0100	XOR	log. exclusive OR	Z,-,-	
5	0101	NOT	log. Negation	Z,-,-	
6	0110	SHR	Shift to right	Z,-,C	LSB → C and ‘0’ → MSB
7	0111	SHL	Shift to left	Z,-,C	MSB → C and ‘0’ → LSB
8	1000	Ø	No operation	-,-,-	Load Register 1 with new data
9	1001	Ø	No operation	-,-,-	Load Register 2 with new data
10	1010	Ø	No operation	-,-,-	Jump Instruction

The Testbench should work synchronously to a clock of 100 MHz and use a high-active reset signal which is active for the first 100ns of the testbench in order to initialize its signals upon startup.

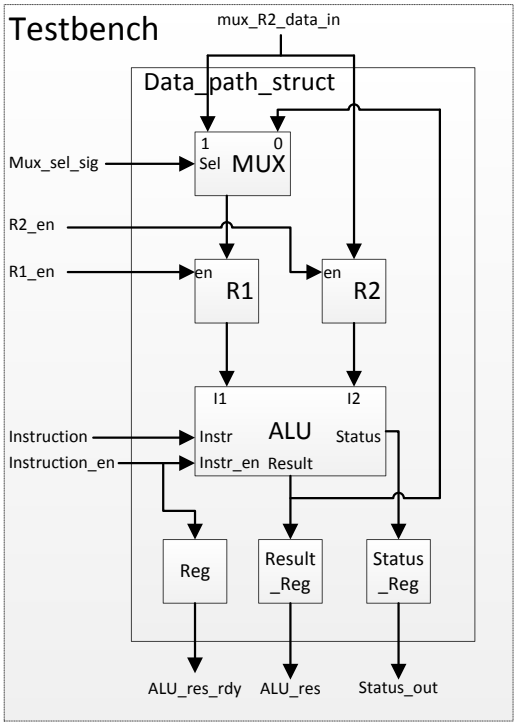
Use following program in order to test the ALU (see also “program.txt”). Store the instructions in an 8-Bit array of the required number of elements (type definition) and iterate through this array during the testbench.

You have the option to use a clocked process for controlling the Testbench or use “wait until rising_edge(…)” / ”wait for …ns” statements within an asynchronous process in order to control the Testbench for driving the ALU signals. Your testbench has to play the role of an opcode interpreter which drives the required signals of the data path in order to load registers, do calculations, perform jumps etc.

Start by writing down a table which contains the expected values for the registers (see Figure) after each instruction of the test program. After implementing the testbench, use a VHDL simulation to check if the design works correctly and if the register values match the expectations.

Additional notes:

- Iterate through the “instruction memory” in order to test the ALU.
- You have to enable the ALU if a computation has to be performed.
- Opcode “1000” and “1001” don’t require processing by the ALU but storing new data to the registers. Opcode “1010” requires modifying the value of the Testbench’s “program counter”.
- Results computed by the ALU need to be stored into register 1.
- You need to control the multiplexer for storing results from the ALU to register 1 or loading new data to the register.



Address	Assembler program	Machine programme	
		OP-Code	Operand
$00	LD1 #C	1000	1100
$01	LD2 #8	1001	1000
$02	ADD	0000	0000
$03	LD2 #4	1001	0100
$04	SUB	0001	0000
$05	SUB	0001	0000
$06	LD1 #C	1000	1100
$07	LD2 #8	1001	1000
$08	JMP #2	1010	0010

Figure 1: ALU structure and assembler program

2.2 Additional Task - Testbench with File Access

Iterating through large data bases in a Testbench which is stored in an array as described above, is inefficient for large amounts of data, like an input image coming from the camera. The behavior of each element of the array needs to be simulated by the Simulator for every simulation step which slows down simulation tremendously.

For that reason, Modelsim provides a library (*std.textio*) for opening files in a Testbench and making available only small amounts of the file at a time. Use the attached template (Rechenwerk_TB_modelsim.vhd) for extending your testbench with the improved input data handling using the *std.textio* library in the **Modelsim simulator**. In this task, the jump instruction doesn't need to be handled for it requires closing and iterating through the file in order to reach the desired address which would require stalling the data_path_struct.

In the "VHDL_Ex2.data\sources_1\hdl" folder you will find a simulation start script which compiles the design files and runs the simulation. Run it by executing *"do run.do"* in the modelsim command line.

Note: The library also allows writing data from the Testbench to a file in order to evaluate the correctness of the simulation results outside of the Modelsim simulation environment (e.g. storing a computed image as a .ppm image file for visualization) – the function calls are *"write(buffer, signal)"*, *"writeline(file, buffer)"*. Don't forget to close an opened file when it is not needed anymore: *"file_close(file)"*.