ti23_assignment_08_Alabrsh_Panov_Zeitler

1a) Implemented N-bit adder and verified correctness by corresponding testbench:
→ siehe src (adder.sv) and src (adder_tb.sv)

1b) Implemented 2:1 multiplexer and verified correctness by corresponding testbench:
→ siehe src (mux_2.sv) and src (mux_2_tb.sv)

1c) Implemented 4:1 multiplexer and verified correctness by corresponding testbench:
→ siehe src (mux_4.sv) and src (mux_4_tb.sv)

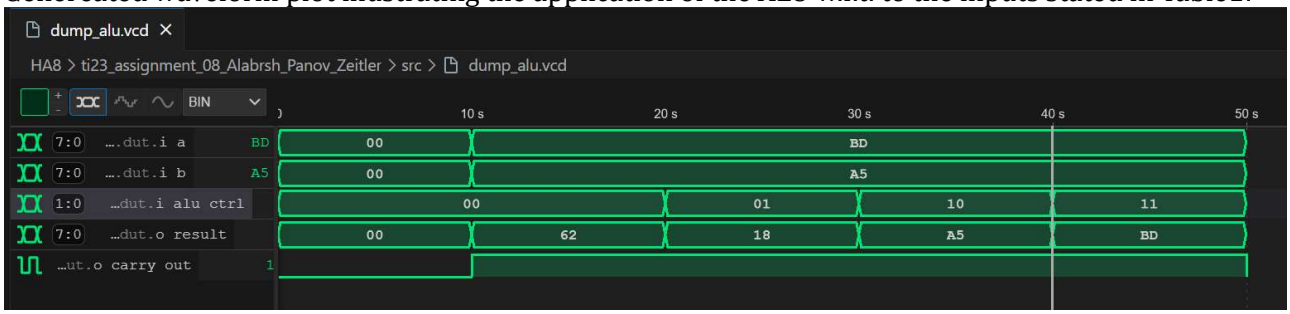
2a) Completed Table1:

i_a	i_b	i_alu_ctrl	o_result	o_carry_out
8'b0000_0000	8'b0000_0000	2'b00	8'b0000_0000	1'b0
8'b1011_1101	8'b1010_0101	2'b00	8'b0110_0010	1'b1
8'b1011_1101	8'b1010_0101	2'b01	**8'b0001_1000**	**1'b1**
8'b1011_1101	8'b1010_0101	2'b10	**8'b1010_0101**	**1'b1**
8'b1011_1101	8'b1010_0101	2'b11	**8'b1011_1101**	**1'b1**

(Example inputs and outputs for basic ALU)

2b) Implemented module Alu using previously programmed MUX and Adder modules and verified correctness by corresponding testbench with tests for all examples in Table1:
→ siehe src (alu.sv) and src (alu_tb.sv)

2c) Genereated waveform plot illustrating the application of the ALU w.r.t. to the inputs stated in Table1:



2d) Implemented top-level module and tested → siehe src (alu_de10_lite.sv) and src (alu_de10_lite_tb.sv)

2e) e) Compile your ALU in Quartus Prime and program the FPGA of a DE10-Lite board. Verify that the boards shows the correct results for the inputs specified in Table 2 and provide a picture of the board for each of the inputs.

i_a	i_b	i_alu_ctrl	o_result	o_carry_out
4'b0011	4'b0010	2'b00	4'b0101	1'b0
4'b1011	4'b1010	2'b01	4'b0001	1'b1
4'b1001	4'b1110	2'b10	4'b1000	1'b1
4'b1001	4'b1110	2'b11	4'b1111	1'b0

Table 2: Example inputs and outputs for a 4-bit ALU when deployed on a DE10-Lite board.

Following Pictures:

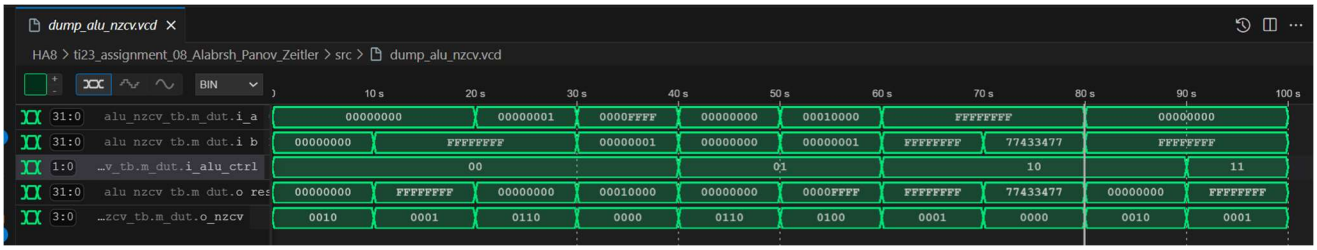
3a) Implemented XOR module and verified correctness by corresponding testbench.
→ siehe src (xor_3.sv) and src (xor_3_tb.sv)

3b) Completed Table3 (Example inputs and outputs for extended ALU):

i_a	i_b	i_alu_ctrl	o_result	o_nzcv
32'h0000_0000	32'h0000_0000	2'b00	32'h0000_0000	4'b0100
32'h0000_0000	32'hffff_ffff	2'b00	32'hffff_ffff	4'b1000
32'h0000_0001	32'hffff_ffff	2'b00	**32'h0000_0000**	4'b0110
32'h0000_ffff	32'h0000_0001	2'b00	**32'h0001_0000**	4'b0000
32'h0000_0000	32'h0000_0000	2'b01	**32'h0000_0000**	4'b0110
32'h0001_0000	32'h0000_0001	2'b01	**32'h0000_ffff**	4'b0010
32'hffff_ffff	32'hffff_ffff	2'b10	**32'hffff_ffff**	4'b1000
32'hffff_ffff	32'h7743_3477	2'b10	**32'h7743_3477**	4'b0000
32'h0000_0000	32'hffff_ffff	2'b10	**32'h0000_0000**	4'b0100
32'h0000_0000	32'hffff_ffff	2'b11	**32'hffff_ffff**	4'b1000

3c) Implemented ALU with Flags module and verified correctness by corresponding testbench with inputs of Table3. → siehe src (alu_nzcv.sv) and src (alu_nzvc_tb.sv)

3d) Genereated waveform plot illustrating the application of the extended ALU to the inputs stated in Table3:



Anmerkung: die Anordnung der NZCV's in der Tabelle ist eine andere Anordnung als die im Template geforderte Anordnung für die NZCV's. Deshalb stimmt o_nzcv im Bild nicht mit o_nzcv in der Tabelle überein. Dennoch ist die Ausgabe richtig. Es ist nur vertauscht. Siehe wie folgt:

Anordnung aus Template:

```
// TODO: finish the implementation by determining the flags
// Set the output o_nzcv such that
// o_nzcv[0] indicates a negative result,
// o_nzcv[1] indicates a zero result,
// o_nzcv[2] indicates a carry over, and
// o_nzcv[3] indicates an overflow.
```

Anordnung von Tabelle: o_nzcv [3] → Negative Flag
o_nzcv [2] → Zero Flag
o_nzcv [1] → Carry Flag
o_nzcv [0] → oVerflow Flag

Aufgabenbearbeitung:

Aufgabe 1 → Rahaf, Christan, Cora
Aufgabe 2 → Rahaf, Christan, Cora
Aufgave 3 → Rahaf, Christan, Cora