Laboration 4

Digitalteknik med VHDL
CPU

Najeem Lawal, David Krapohl

ver.:February 11, 2022

1 Component reuse

In this lab we will re-use existing components from previous labs, components developed by other engineers and finally create ONLY ONE component called CPU. Emphasis now is on RESPECTING existing, functional and verified components. By RESPECTING we mean DO NOT CHANGE EXISTING COMPONENTS. Implement your new component based on specification and integrate with other components are implemented according to specifications. Feel free to create a test-bench and simulate your new design.

1.1 Task 1: Designing the CPU

Create a VHDL module according to fig. 1 that implements a CPU entity shown in listing in section 1.1:

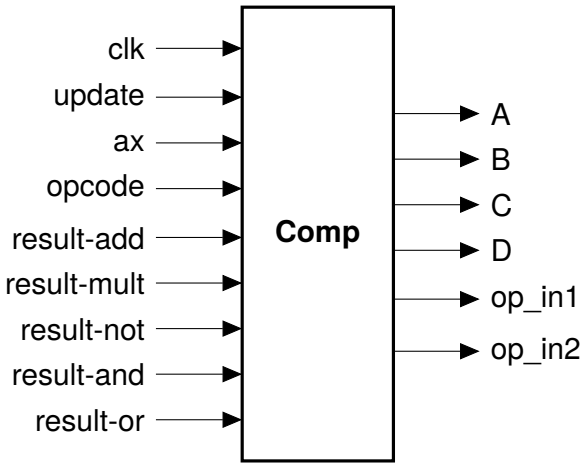

Figure 1: Entity of Comp

```
entity cpu is
    Port (
        clk     : in STD_LOGIC;
        update      : in STD_LOGIC;
```

Table 1: CPU instruction set

Opcode	Instruction	Description	Comment
0000	MOV AX BX	BX <= AX	Assigns to BX
0001	MOV AX CX	CX <= AX	Assigns to CX
0010	MOV AX DX	DX <= AX	Assigns to DX
0011	MOV BX "0000"	BX <= "0000"	Reset BX
0100	MOV CX "0000"	CX <= "0000"	Reset CX
0101	MOV DX "0000"	DX <= "0000"	Reset DX
0110	ADD "0010" "0011" CX	CX <= 2+3	Test adder
	MOV DX "0101"	DX <= "0101"	
0111	MUL "0010" "0011" CX	CX <= $2 \cdot 3$	Test multiplier
	MOV DX "0110"	DX <= "0110"	Expected value
1000	INV "0010" CX	CX <= NOT("0010")	Test Inverter
	MOV DX "1101"	DX <= "1101"	Expected value
1001	AD4 "0010" "0011" CX	CX <= "0010" AND "0011"	Test AND
	MOV DX "0010"	DX <= "0010"	
1010	OR4 "0010" "0011" CX	CX <= "0010" OR "0011"	Test OR
	MOV DX "0011"	DX <= "0011"	Expected value
1011	ADD AX BX [DX&CX]	DX <= "000"&(bit4), CX <= LSB4	AX + BX
1100	MUL AX BX [DX&CX]	DX <= MSB4, CX <= LSB4	AX · BX
1101	INV AX CX	CX <= NOT(AX)	Invert AX
	MOV DX "0000"	DX <= "0000"	
1110	AD4 AX BX CX	CX <= AX AND BX	AX AND BX
	MOV DX "0000"	DX <= "0000"	
1111	OR4 AX BX CX	CX <= AX OR BX	AX OR BX
	MOV DX "0000"	DX <= "0000"	

```
        ax          : in STD_LOGIC_VECTOR(3 downto 0);
        opcode      : in STD_LOGIC_VECTOR(3 downto 0);
        result_add  : in STD_LOGIC_VECTOR(4 downto 0);
        result_mult : in STD_LOGIC_VECTOR(7 downto 0);
        result_not  : in STD_LOGIC_VECTOR(3 downto 0);
        result_and  : in STD_LOGIC_VECTOR(3 downto 0);
        result_or   : in STD_LOGIC_VECTOR(3 downto 0);
        A           : out STD_LOGIC_VECTOR(3 downto 0);
        B           : out STD_LOGIC_VECTOR(3 downto 0);
        C           : out STD_LOGIC_VECTOR(3 downto 0);
        D           : out STD_LOGIC_VECTOR(3 downto 0);
        op_in1      : out STD_LOGIC_VECTOR(3 downto 0);
        op_in2      : out STD_LOGIC_VECTOR(3 downto 0)  );
end cpu;
```

The CPU operates by analysing the value on the input port opcode to determine how to process the input port `AX` as well as how to determine the value of the output ports A, B, C and D The table below describes the behaviour of the CPU based on the opcode. The behaviour of the CPU should be implemented using CASE-WHEN states in VHDL.

The CPU need only three internal signals `BX`, `CX` and `DX`. `AX`, `BX`, `CX` and `DX` should be mapped to the output ports A, B, C and D respectively at the architecture level. Inside a clocked process, at the rising edge of the clock determine the value of `BX`, `CX` and `DX` base opcode in the CPU operations table when the update input port is high. From opcode `0110` to `1111` the data or signal to be assigned to output ports input1 and input2 are specified. the ports will be mapped in parallel to the five 4-bit ALU operators in the higher top module. The output of the 4-bit ALU operators are read through the specified port. These ports should be appropriately mapped to `CX` and `DX` based on the CPU operations table. When `opcode=1011`, addition result is 5 bits. Concatenate three 0's in front of the bit 4 and assign them to DX whereas the lower 4 bits of the adder are assigned to CX. Also when `opcode=1100`, multiplication the result becomes 8 bits. The 4 most significant bits should be stored in DX and the 4 least significant bits should be stored in CX.

1.2 Task2: CPU Simulation

Create a testbench and simulate the CPU to verify that it behaves correctly.

2 Report

Write a report focusing on tasks 1 and 2. The report should include VHDL-code as text (not as images) and understandable results from simulations (as screenshot not as phone snapshot). Submit your report in the Moodle page according to the deadline.