Übung 3

In dieser und den folgenden Übungen soll ein einfacher RISC-Prozessor mit Load/Store-Architektur entworfen, auf ein FPGA abgebildet und anschließend programmiert werden.

Prozessorspezifikation

Der Prozessor soll folgenden Befehlssatz unterstützen:

Befehl	***Opcode***	***Bedeutung***
LD	01 0-- xxx --- zzz --	Rxxx ← M[Rzzz]
ST	01 1-- --- yyy zzz --	M[Rzzz] ← Ryyy
LDIH	10 0-- xxx yyy zzz cc	Rxxx[15:8] ← yyyzzzcc
LDIL	10 1-- xxx yyy zzz cc	Rxxx[7:0] ← yyyzzzcc
JMP	11 00- --- yyy --- --	pc ← Ryyy
JZ	11 01- --- yyy zzz --	pc ← Ryyy if Rzzz==0
CALL	11 11- xxx yyy --- --	Rxxx ← pc+1, pc ← Ryyy
NOP	11 10- --- --- --- --	Keine Operation
ADD	00 000 xxx yyy zzz --	Rxxx ← Ryyy +Rzzz
SUB	00 001 xxx yyy zzz --	Rxxx ← Ryyy -Rzzz
AND	00 010 xxx yyy zzz --	Rxxx ← Ryyy AND Rzzz
OR	00 011 xxx yyy zzz --	Rxxx ← Ryyy OR Rzzz
CP	00 100 xxx --- zzz --	Rxxx ← Rzzz
NOT	00 101 xxx --- zzz --	Rxxx ← NOT Rzzz
SAL	00 110 xxx --- zzz --	Rxxx ← Rzzz << 1
SAR	00 111 xxx --- zzz --	Rxxx ← Rzzz >>1, Rxxx[15] ← Rzzz[15]

Tabelle 4: Befehlssatz

Der Prozessor soll gemäß folgendem Blockschaltbild aufgebaut werden. In den verschiedenen Übungen werden die einzelnen Blöcke entworfen und am Ende zum Gesamtsystem zusammengefasst.

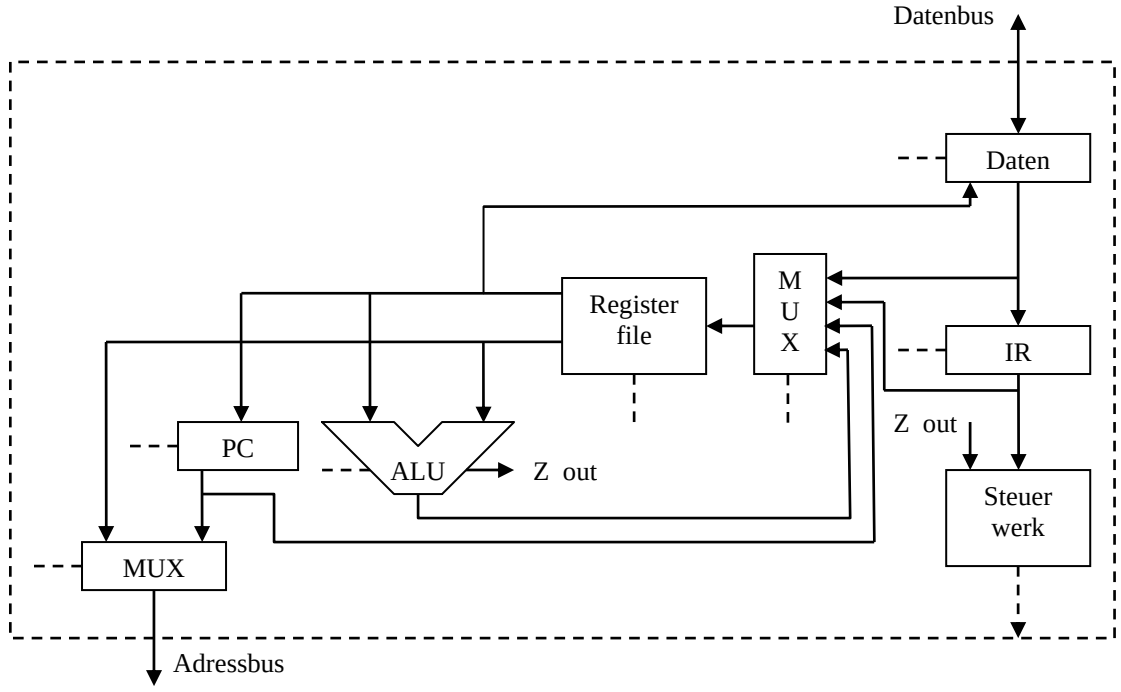

Abbildung 9: Blockschaltbild des Prozessors

Merkmale des Prozessors:

- Load-Store-Architektur
- Triadischer Befehlssatz (OpCode destReg, srcReg0, srcReg1)
- 8 Register r0, …, r7
- 16 Bit Daten (Zweierkomplementdarstellung)
- 16 Bit Adressen

Die ALU

Als Erstes entwerfen wir die ALU. Sie soll Daten in 16 Bit Zweierkomplementdarstellung verarbeiten und folgende Operationen zur Verfügung stellen:

- Addition ADD
- Subtraktion SUB
- Bitweise Konjunktion AND
- Bitweise Disjunktion OR
- Bitweise Negation NOT
- Bitweise Identität CP
- Arithmetisches Links-Schieben SAL
- Arithmetisches Rechts-Schieben SAR
- Vergleich mit 0 setzt Nullsignal Z_OUT bei Rzzz=0

Der Entwurf soll hierarchisch aus 4-Bit-ALUs zusammengesetzt werden:

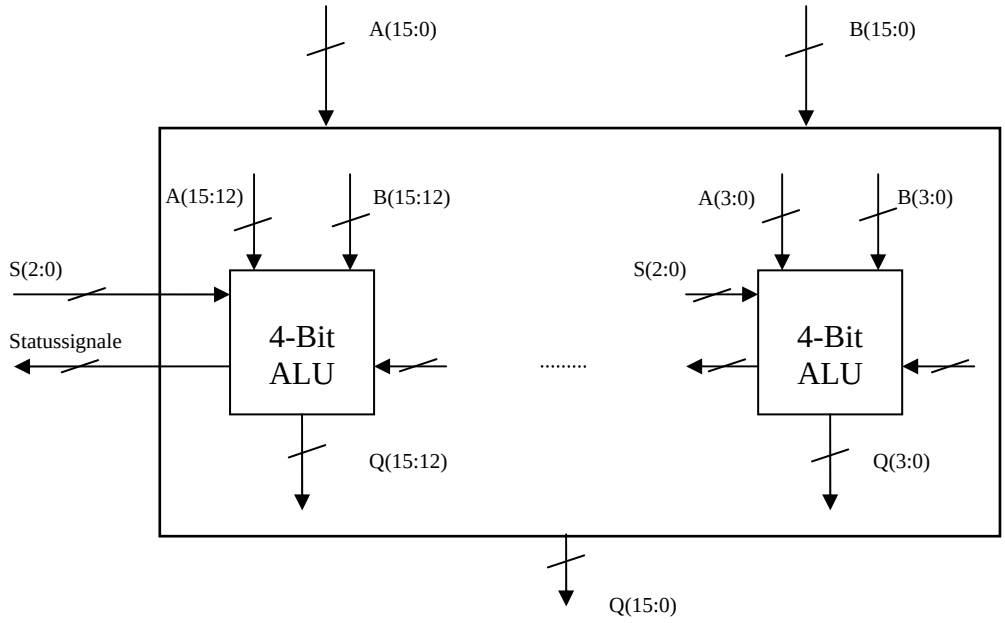

Abbildung 10: Hierarchischer Aufbau der ALU

Die Steuersignale für die ALU werden an jede einzelne 4-Bit ALU weitergereicht, während die Verbindungen zwischen den Blöcken z.B. Carry-Bits weitergeben oder für die Implementierung der Schiebebefehle benutzt werden. Statussignale (z.B. Nullanzeige Z_OUT) werden aus dem letzten Block abgeleitet.

Entwurf der 4-Bit-ALU

Das Bild zeigt eine 4-Bit-ALU im Detail, die Tabelle liefert die genaue Schnittstellendefinition. Erstellen Sie die Spezifikation für den Entwurf der Basiszelle, d.h. eine Tabelle mit Belegung der Steuereingänge und den entsprechenden Operationen. Das Nullsignal Z_OUT soll genau dann den Wert 1 annehmen, wenn B(3:0)=0000 und Z_IN=1 gilt.

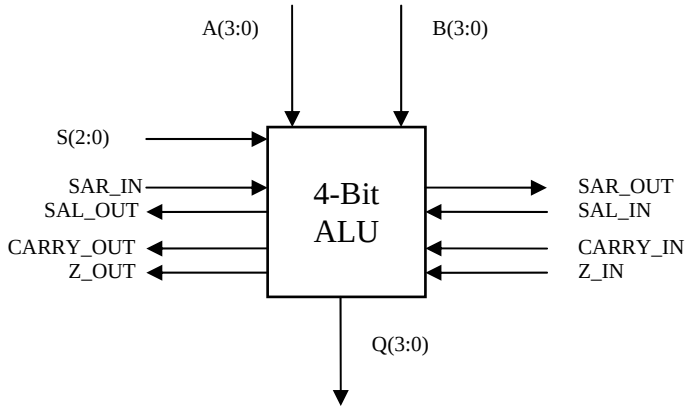

Abbildung 11: 4-Bit-ALU

Port	***Richtung***	***Funktion***
S(2:0)	IN	Steuereingänge
A(3:0), B(3:0)	IN	Dateneingänge
SAL_IN, SAR_IN	IN	Eingänge für Schiebebefehle
CARRY_IN, Z_IN	IN	Carry- und Zero-Eingang
Q(3:0)	OUT	Datenausgang
SAR_OUT, SAL_OUT	OUT	Ausgänge für Schiebebefehle
CARRY_OUT, Z_OUT	OUT	Carry- und Zero-Ausgang

Tabelle 5: Portbeschreibung entity ALU4

Wählen Sie für die Belegung der Steuereingänge S(2:0) die drei signifikanten Bits des Entsprechenden Op-Codes. Für die einstelligen Operationen CP, NOT, SAL und SAR soll der Dateneingang B verwendet werden. Dies erleichtert später wesentlich den Entwurf!

Verhaltensbeschreibung

Erstellen Sie ein VHDL-Modell ihrer Spezifikation:

- Erstellen Sie mit ActiveHDL ein neues Design mit dem Namen **ALU**
- Erstellen Sie eine neue VHDL-Datei *alu4_spec.vhd* und definieren Sie darin eine **entity ALU4** mit einer dazugehörigen **architecture SPEC**, welche das Verhalten der 4-Bit-ALU beschreibt.

Wenn Sie zur Modellierung der arithmetischen Funktionen den Datentyp `integer` verwenden wollen, können Sie die untenstehenden Konvertierungsfunktionen, die in der IEEE Bibliothek im Paket *IEEE.std_logic_arith* enthalten sind, verwenden.

```
function CONV_INTEGER (ARG: STD_LOGIC_VECTOR) return INTEGER;
function CONV_STD_LOGIC_VECTOR (ARG: INTEGER; SIZE: INTEGER) return STD_LOGIC_VECTOR;
```

Die Funktion conv_integer interpretiert den std_logic_vector ARG als Binärzahl und liefert einen Integerwert zurück.
Die Funktion conv_std_logic_vector liefert die size niederwertigsten Bits des Integerwertes als std_logic_vector.

Im Folgenden nun ein Ausschnitt aus der entsprechenden VHDL-Beschreibung:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity ALU4 is
      port( A,B: in std_logic_vector(3 downto 0);
            CARRY_IN,Z_IN,SAL_IN,SAR_IN: in std_logic;
            S: in std_logic_vector (2 downto 0);
            CARRY_OUT,Z_OUT,SAL_OUT,SAR_OUT: out std_logic;
            Q: out std_logic_vector (3 downto 0));
end entity ALU4;

architecture SPEC of ALU4 is
begin
      operate_alu: process (S,A,B,CARRY_IN,Z_IN,SAL_IN,SAR_IN)
      ...
      begin
            ...
            case S is

                  ...
                  -- and
                  when "010" =>
                  Q<=(A and B);
                  ...
                  when others => NULL;
            end case;
      end process operate_alu;
end architecture SPEC;
```

- Simulieren und testen Sie die Verhaltensbeschreibung. Wählen Sie dazu ALU4 als top-level-unit und überprüfen Sie die Funktion in einem Signal-Fenster, wobei sie an den Signaleingängen von ihnen gewählte Testpatterns mit *Waveform->Stimulators* anlegen und die Ergebnisse überprüfen. Beseitigen Sie detektierte Fehler.

Entwurf auf Gatterebene

- Erstellen Sie nun eine Gatternetzliste für Ihre 4-Bit ALU, wobei Ihnen folgende Gatter zur Verfügung stehen:

INV	Inverter
AND2	AND-Gatter mit 2 Eingängen
NAND2	NAND-Gatter mit 2 Eingängen
OR2	OR-Gatter mit 2 Eingängen
NOR2	NOR-Gatter mit 2 Eingängen
XOR2	XOR-Gatter mit 2 Eingängen
XNOR2	XNOR-Gatter mit 2 Eingängen
MUX2X1	2:1 Multiplexer
VA	Volladdierer

- Geben Sie Ihre Gatternetzliste als Blockdiagramm in ActiveHDL ein. Gehen Sie dazu im Design-Browser auf die vorhin eingegebene **entity ALU4** und fügen Sie über das Kontextmenü mit *Add new Architecture* eine neue **architecture impl** als Blockdiagramm an (*alu4_impl.bde*). Verwenden Sie die Gatter aus den *Built in Symbols* in der *Symbols Toolbox*.
- Um den Volladdierer aus der 1. Übungseinheit nutzen zu können, fügen Sie dem Projekt die entsprechenden Dateien hinzu (im Menü *Design->Add Files to Design*) und übersetzen Sie diese. Sodann steht Ihnen in der *Symbols Toolbox* ein Symbol für Ihren Volladdierer zur Verfügung.