Primjer: **Pojednostavljeni floating-point sabirač**

Floating-point je format zapisa broja. Istim brojem bita, opseg brojeva zapisanih u floating-point formatu je mnogo širi nego u signed integer formatu.
U okviru primjera potrebno je realizovati 13-bitni pojednostavljeni floating-point sabirač, ne uzimajući u obzir grešku do koje dolazi usljed zaokruživanja. Reprezentacija broja je sljedeća:

$$(-1)^{s} \times f \times 2^{e}$$

- sign bit **s**: ima vrijednost '1' ukoliko je broj negativan, odnosno '0' ukoliko je broj pozitivan,
- 4-bitni eksponent **e,**
- 8-bitna mantisa (significand, fraction) **f**.

Uvedene su sljedeće pretpostavke:
- Eksponent i mantisa su zapisani u unsigned formatu,
- Reprezenatcija mora biti normalizovana ili jednaka nuli. Normalizovana reprezentacija znači da MSB mora biti 1. Ukoliko je magnituda rezultata sabiranja manja od najmanje nenulte vrijednosti koju je moguće zapisati na ovaj način ($0.10000000 \times 2^{0000}$), potrebno je konvertovati u nulu.

Uzimajući u obzir prethodne pretpostavke, najveća i najmanja nenulta magnituda su:

$$\max = 0.11111111 \times 2^{1111}$$

i

$$\min = 0.10000000 \times 2^{0000},$$

što znači da je opseg vrijednosti oko 2^{16}.

Procedura sabiranja dva broja je sljedeća:
- Sortiranje. Upoređivanje apsolutnih vrijednosti brojeva kako bi se odredio veći broj,
- Izjednačavanje eksponenata. Svođenje na eksponent većeg broja, tako što se mantisa pomjera udesno za broj mjesta jednak razlici u eksponentima,
- Sabiranje/oduzimanje mantisa brojeva,
- Normalizacija rezultata.

Normalizacija rezultata podrazumijeva:
- Nakon oduzimanja, rezultat može sadržati nule na pozicijama viših bita u okviru mantise,
- Nakon oduzimanja, rezultat može imati suviše malu vrijednost, pa ga je potrebno konvertovati u nulu (tabela 1, drugi primjer),
- Nakon sabiranja, rezultat može generisati carry-out bit (tabela 1, četvrti primjer).

U tabeli 1 dato je nekoliko primjera sabiranja dva broja u floating-point zapisu, pri čemu je preciznost zapisa mantise dvije cifre, dok je za eksponent rezervisana jedna cifra.

U okviru primjera, koristiće se prethodno opisan algoritam. Sufiksi 'b', 's', 'a', 'r' i 'n' označavaju "veći broj", "manji broj", "poravnate brojeve (nakon izjednačavanja eksponenata)", "rezultat" i "normalizovan rezultat", respektivno.

Tabela 1 – Floating-point sabiranje, primjeri

primjer	sort	align	add/sub	normalize
+0,54e3 -0,87e4	-0,87e4 +0,54e4	-0,87e4 +0,54e4	-0,87e4 +0,54e4 -0,82e4	-0,82e4
+0,54e3 -0,55e3	-0,55e3 +0,54e3	-0,55e3 +0,54e3	-0,55e3 +0,54e3 -0,01e3	-0,10e2
+0,54e0 -0,55e0	-0,55e0 +0,54e0	-0,55e0 +0,54e0	-0,55e0 +0,54e0 -0,01e0	-0,00e0
+0,56e3 +0,52e3	+0,56e3 +0,52e3	+0,56e3 +0,52e3	+0,56e3 +0,52e3 +1,07e3	+0,1e4

Rješenje

VHDL kod moguće realizacije pojednostavljenog floating-point sabirača dat je u listingu 1.

Listing 1 – Pojednostavljeni floating-point sabirač

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fp_adder is
	port
		(
			sign1, sign2 : in std_logic;
			exp1, exp2 : in std_logic_vector(3 downto 0);
			frac1, frac2 : in std_logic_vector(7 downto 0);
			sign_out : out std_logic;
			exp_out : out std_logic_vector(3 downto 0);
			frac_out : out std_logic_vector(7 downto 0)
		);

end fp_adder;

architecture arch of fp_adder is

	signal signb, signs : std_logic;
	signal expb, exps, expn : unsigned(3 downto 0);
	signal fracb, fracs, fraca, fracn : unsigned(7 downto 0);
	signal sum_norm : unsigned(7 downto 0);
	signal exp_diff : unsigned(3 downto 0);
	signal sum : unsigned(8 downto 0);
	signal lead0 : unsigned(2 downto 0);

begin

-- sortiranje --

	process(sign1, sign2, exp1, exp2, frac1, frac2)
	begin
		if(exp1 & frac1) > (exp2 & frac2) then
			signb <= sign1;
			signs <= sign2;
			expb <= unsigned(exp1);
```

```
                  exps <= unsigned(exp2);
                  fracb <= unsigned(frac1);
                  fracs <= unsigned(frac2);
            else
                  signb <= sign2;
                  signs <= sign1;
                  expb <= unsigned(exp2);
                  exps <= unsigned(exp1);
                  fracb <= unsigned(frac2);
                  fracs <= unsigned(frac1);
            end if;
      end process;

-- izjednacavanje eksponenata --

      exp_diff <= expb - exps;
      with exp_diff select
            fraca <= fracs                                  when "0000",
                     "0"       & fracs(7 downto 1)          when "0001",
                     "00"      & fracs(7 downto 2)          when "0010",
                     "000"     & fracs(7 downto 3)          when "0011",
                     "0000"    & fracs(7 downto 4)          when "0100",
                     "00000"   & fracs(7 downto 5)          when "0101",
                     "000000"  & fracs(7 downto 6)          when "0110",
                     "0000000" & fracs(7)                   when "0111",
                     "00000000"                             when others;

-- sabiranje/oduzimanje --

      sum <= ('0' & fracb) + ('0' & fraca) when signb = signs else
                  ('0' & fracb) - ('0' & fraca);
-- normalizacija rezultata --

-- brojanje vodecih nula mantise --

      lead0 <= "000" when (sum(7) = '1') else
               "001" when (sum(6) = '1') else
               "010" when (sum(5) = '1') else
               "011" when (sum(4) = '1') else
               "100" when (sum(3) = '1') else
               "101" when (sum(2) = '1') else
               "110" when (sum(1) = '1') else
               "111";

-- siftovanje mantise ulijevo --

      with lead0 select
            sum_norm <=
                     sum(7 downto 0)                        when "000",
                     sum(6 downto 0) & '0'                  when "001",
                     sum(5 downto 0) & "00"                 when "010",
                     sum(4 downto 0) & "000"                when "011",
                     sum(3 downto 0) & "0000"               when "100",
                     sum(2 downto 0) & "00000"              when "101",
                     sum(1 downto 0) & "000000"             when "110",
                     sum(0)          & "0000000"            when others;

      process(sum, sum_norm,expb,lead0)
      begin
            if sum(8) = '1' then -- carry out, siftuj mantisu udesno
                  expn <= expb + 1;
                  fracn <= sum(8 downto 1);
            elsif lead0 > expb then -- mala vrijednost za normalizaciju
                  expn <= (others => '0');
                  fracn <= (others => '0');
```

```
            else
                  expn <= expb - lead0;
                  fracn <= sum_norm;
            end if;
      end process;

      sign_out <= signb;
      exp_out <= std_logic_vector(expn);
      frac_out <= std_logic_vector(fracn);

end arch;
```

Kreirati VHDL **testbench**, pri čemu je ulazne signale potrebno učitati iz odgovarajućeg fajla (pogledati vježbe 4). Izvršiti simulaciju rada kola. Prikazati rezultate simulacije.

U cilju verifikacije rada sistema, realizovati floating-point sabirač kod koga su za eksponent rezervisana 3 bita, a za mantisu 4 bita. Vrijednost eksponenta unaprijed zadati kao konstantu, dok je vrijednost mantise jednaka kod oba broja i zadaje se preko prekidača na razvojnoj platformi. Rezultat sabiranja prikazati pomoću LED-a.

Izvršiti procese **synthesize**, **translate**, **map** i **place** & **route**.

Izvršiti implementaciju kola uz pomoć **Spartan-3E Starter Kit** razvojne platforme (pogledati uputstvo u okviru vježbi 2) i verifikovati rad kola.