A2CPU - jednostavni 8-bitni procesor

Tehnička dokumentacija
Verzija 1.0

Student : Mijo Tvrdojević

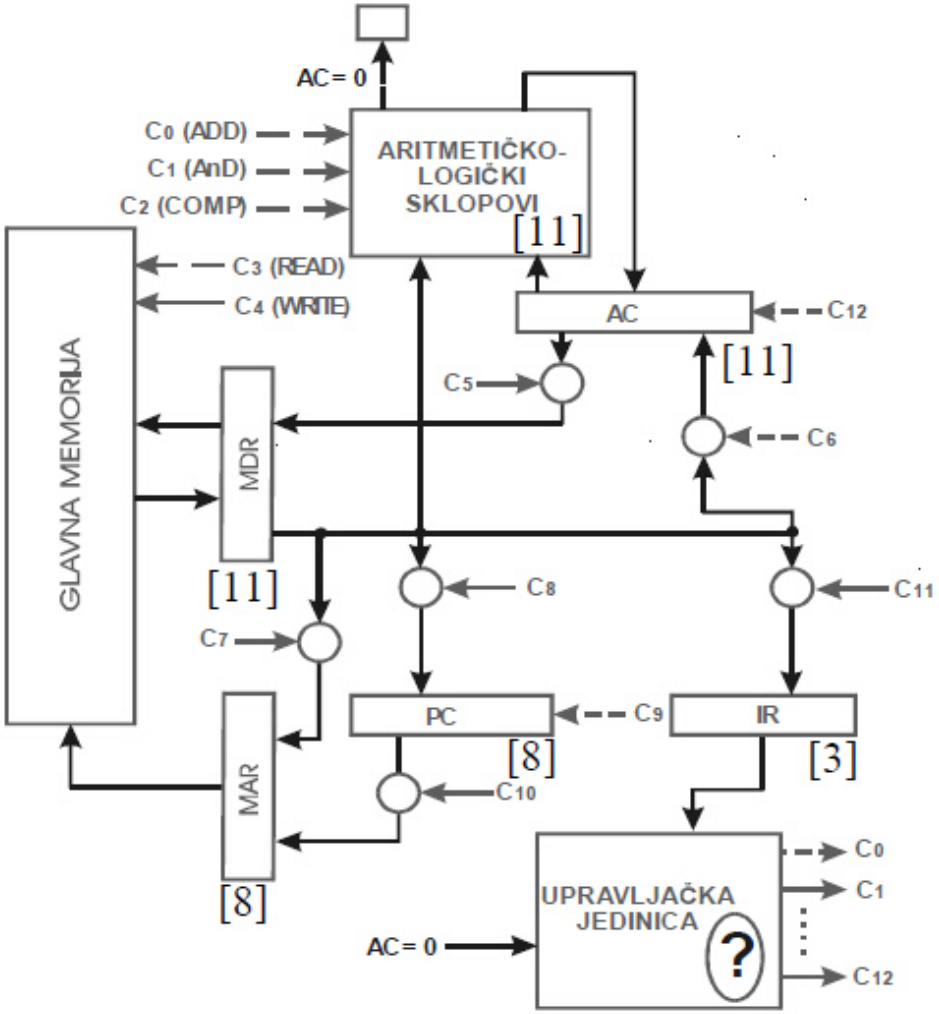
Nastavnik: prof.dr.sc Siniša Šegvić

Predmet : Arhitektura računala 2

Sadržaj

1. Uvod

Procesor razvijen u ovom projektu razmatran je na predavanjima u okviru kolegija Arhitektura računala 2, FER Zagreb 2009. Radi se o 8-bitnom procesoru pojednostavljene arhitekture, u svrhu lakšeg razumijevanja osnovnih principa njegovog rada. Na sljedećoj je slici shema njegove arhitekture:



Na slici su naznačene širine sabirnica. Cijela arhitektura je maksimalno pojednostavljena, već prema formatu instrukcija . Obilježja arhitekture instrukcija (ISA) ovog procesora su :

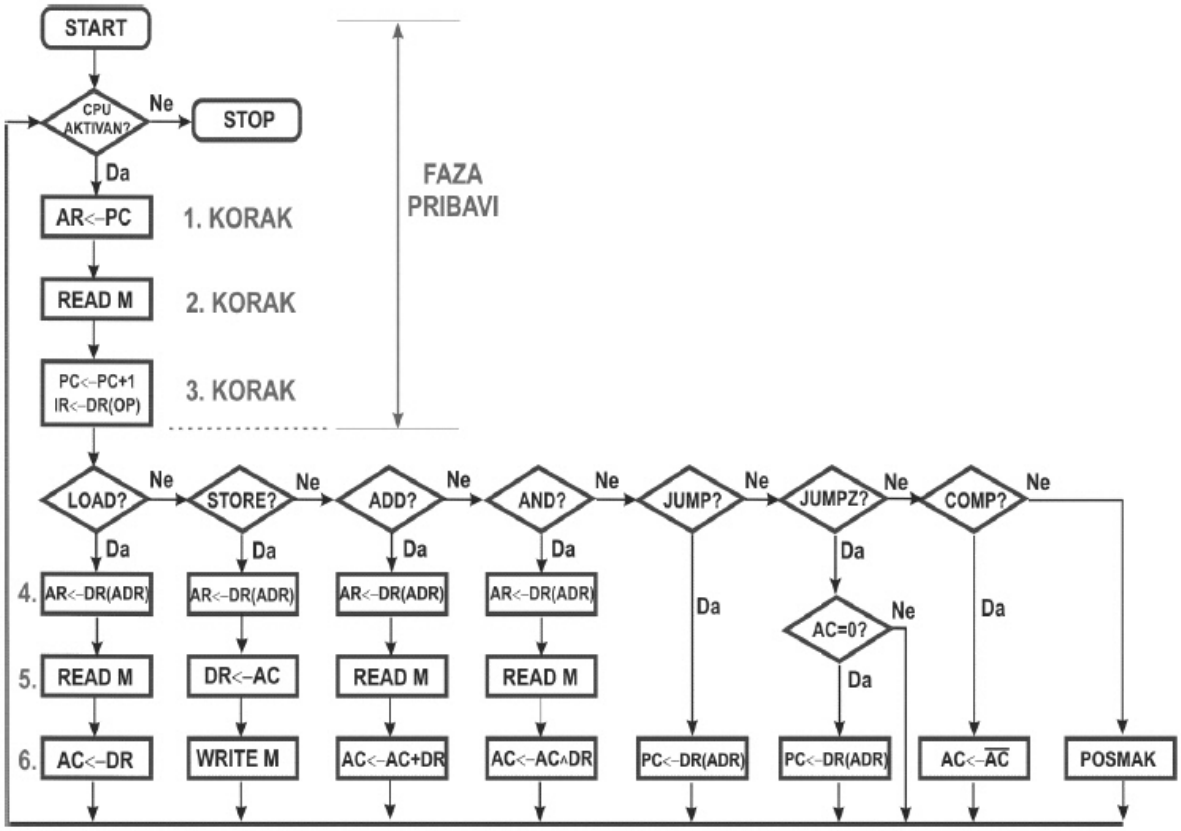
- postoji 8 instrukcija : ld, st, add, and, jmp, jmpz, comp, shr :

Mnemonik	**Opis**
ld X ; load X	AC ← M(X)
st X ; store X	M(X) ← AC
add X	AC ← AC + M(X)
and X	AC ← AC ∧ M(X)
jmp X ; jump X	PC ← X
jmpz X ; jump if zero	if AC = 0 then PC ← X
not ; bitwise negation	AC ← ¬ AC
rsh ; shift right	posmak AC udesno

- svaka zauzima jednu riječ memorije, koja je u ovoj arhitekturi 11- bitna
- svaka se instrukcija sastoji iz operacijskog koda (bitovi 10 do 8) i adrese operanda u izravnom adresiranju (bitovi 7 do 0)
- adresna sabirnica stoga je širine 8, podatkovna 11 bitova

Svaka pojedina instrukcija se temelji na mikroinstrukcijama koje su nedjeljive i izravno sklopovski podržane. Izvođene instrukcije se odvija u 2 faze : fazi pribavljanja i fazi izvršavanja. Faza dohvata za sve instrukcije traje jednako: 4 ciklusa takta procesora. Faza izvršavanja traje 1 ili 4 takta, ovisno o kojoj je instrukciji riječ. Naime, instrukcije koje moraju dohvaćati operand iz memorije traju 4 ciklusa zbog dodatnog pristupa memoriji, dok ostale traju samo 1 ciklus.

Na slijedećoj slici prikazan je dijagram izvršavanja instrukcija:



Vidimo da se svaka instrukcija sastoji od mikroinstrukcija. Sve te mikroinstrukcije traju 1 ciklus takta, osim memorijskih mikroinstrukcija READ i WRITE za koje se u ovom modelu pretpostavlja trajanje od 2 ciklusa takta. Izvođenje svake mikroinstrukcije kontrolira upravljačka jedinica pomoću kontrolnih bitova. Svaki kontrolni bit je vezan za jednu mikroinstrukciju i određuje hoće li se ona izvršiti u trenutnom taktu: ako je upravljački signal u logičkoj jedinici onda se mikroinstrukcija izvršava, inače ne. Popis signala i mikroinstrukcija koje kontroliraju dan je slikom:

signal	mikrooperacija	
C_0	AC ← AC + DR	
C_1	AC ← AC ∧ DR	
C_2	AC ← ¬AC	
C_3	MDR ← M(MAR)	/READ M/
C_4	M(MAR) ← MDR	/ WRITE M/
C_5	MDR ← AC	
C_6	AC ← MDR	
C_7	MAR ← MDR(A)	
C_8	PC ← MDR(A)	
C_9	PC ← PC + 1	
C_{10}	MAR ← PC	
C_{11}	IR ← MDR(I)	
C_{12}	AC ← AC >> 1	

Upravljačka jedinica procesora je dakle zadužena za generiranje upravljačkih signala. Ulaz u upravljačku jedinicu su instrukcija i signal takta, a izlaz upravljački signali. Stoga su dakle upravljački signali logička funkcija instrukcije i trenutnog takta, odnosno:

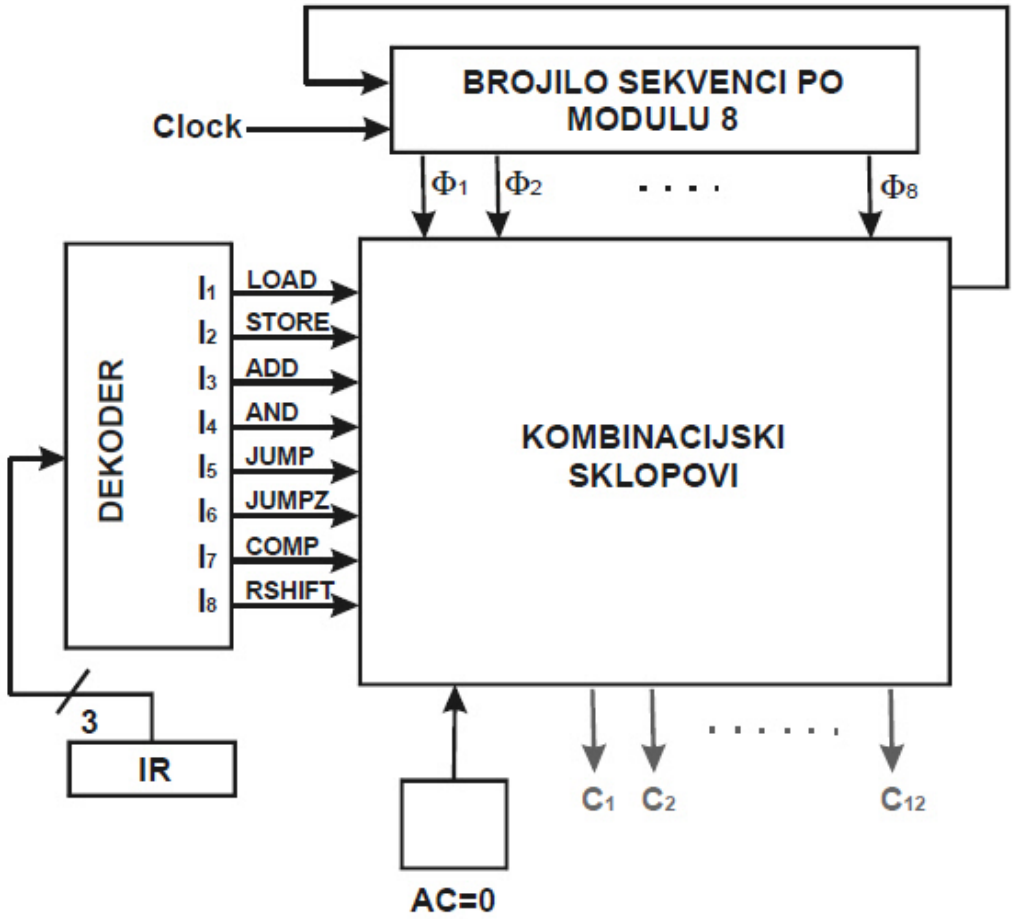
$$C_i = \sum_j \left(\Phi_j \bullet \sum_m I_m \right)$$

2. VHDL izvedba procesora i sinteza u Xilinx-u

Za opis strukture procesora korišten je jezik VHDL. Treba napomenuti da je svaka komponenta pravljena generički, odnosno prilikom instanciranja svake može se navesti širina sabirnica koje su vezane za tu komponentu. Iako ovo ima smisla samo za podatkovnu sabirnicu (promjena širine ostalih sabirnica iziskuje i promjenu arhitekture procesora), na ovaj način se povećava čitljivost koda i njegova ponova uporaba.

2.1 Upravljačka jedinica

Upravljačka se jedinica sastoji iz tri dijela : instrukcijskog dekodera, brojača sekvenci i kombinacijskog polja. To se vidi na slijedećoj shemi :



Izlaz su instrukcijski kod iz instrukcijskog registra, te signal takta (clock).

2.1.1 Dekoder instrukcija

Dekoder instrukcija prima na ulazu 3-bitni kod instrukcije, a na izlazu postavlja bit čija pozicija odgovara dekadskom kodu instrukcije u 1 a ostale u 0. Izlaz je stoga širine 8 bitova. VHDL implementacija je jednostavna, i dana u slijedećem listingu:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity InstructionDecoder is
  generic(
    iw_g: integer :=3;    --širina instrukcije
    ow_g: integer :=8     --širina dekodiranog signala
  );
  port(
    dekbit_p : in std_logic_vector(iw_g-1 downto 0);
    instout_p : out std_logic_vector(ow_g -1 downto 0)
    );
end InstructionDecoder;

architecture Behavioral of InstructionDecoder is

begin

  process(dekbit_p)
    variable input_v : integer;
  begin
    input_v := conv_integer(dekbit_p);
    instout_p <= ( others => '0');
    instout_p(input_v) <= '1';
  end process;

end Behavioral;
```

Kod je sasvim jasan: signal na ulazu (tipa std_logic_vector, niz logičkih vrijednosti) se pretvara pomoću conv_integer u dekadsku vrijednost i sprema u variablu input_v. Nakon toga se bit na mjestu input_v izlaza postavlja u 1 a ostali u 0.

2.1.2 Brojilo sekvenci

Ovaj sklop služi kao brojač signala takta modulo 5 odnosno 8. To znači da broji od 0 do 4 ili 7, a nakon što odbroji do kraja, ponovo kreće od nule. Tako radi upravo zbog instrukcijskog skupa: svaka instrukcija se sastoji od mikroinstrukcija koje se izvršavaju u određenim trenutcima izvođenja instrukcije. Upravo te trenutke određuje brojilo, a budući da memorijske instrukcije traju 8 ciklusa, a nememorijske 5, broji od 0 do 7 odnosno od 0 do 4. Za ove potonje se dakle izvodi kraćenje trajanja brojanja, i to tako da se generira reset signal koji postavlja brojilo ponovo u početno stanje (dakle prije nego odbroji do 7).

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

entity SequenceCounter_v2 is
  generic(
    pw_g: integer :=8
  );
  port (
    reset_counter_p : in std_logic;  --glavni reset signal
    reset_short_p : in std_logic; --reset brojila na 0 kad se krati ciklus
    start_p : in std_logic;
    stop_p : in std_logic;
    clock_p : in std_logic;
    phi_p : out std_logic_vector(0 to pw_g-1)
  );
end SequenceCounter_v2;

architecture Behavioral of SequenceCounter_v2 is
  signal running_s : boolean;
  signal phi_s : std_logic_vector(0 to pw_g-1);
  signal reseted_s : boolean := false;
begin
```

```
  process(start_p, stop_p, clock_p,reset_counter_p) begin
    if reset_counter_p = '1' then --glavni reset
      phi_s <= (others => '0'); --postavi izlaz u 0
      reseted_s <= true; -- brojilo je resetirano
      running_s <= false; --zaustavi brojanje ako je postojalo
    elsif stop_p = '1' then
      running_s <= false;
    elsif start_p ='1' then
      running_s <= true;
    elsif rising_edge(clock_p) and running_s then
      if reset_short_p = '1' then --vrati se na 0 ako je došlo do kraćenja
            phi_s <=((0) => '1', others => '0');
      else
       if reseted_s then --ako je prije pokretanja resetirano brojilo
        phi_s <= ((0) => '1', others => '0');-- vrati se na 0
        reseted_s <= false;
      --inače, uvećaj izlaz za 1(pomakni bitove udesno za 1)
       else phi_s <= phi_s(pw_g-1) & phi_s(0 to pw_g-1-1);
       end if;
      end if;
    end if;
  end process;

  phi_p <= phi_s;

end Behavioral;
```

Brojilo ima dodatne ulaze stop i start, za zaustavljanje i pokretanje procesora. Glavni reset resetira cijelo brojilo i zaustavlja ga (running_s se postavlja u 0), a brojilo se pokreće ponovo dizanjem signala start_p u 1. Reset prilikom kraćenja ciklusa ne zaustavlja brojilo već ga samo vraća na 0.

2.1.3 Kombinacijski sklop (za generiranje upravljačkih signala)

Generator signala generira upravljačke signale koji upravljaju izvođenjem mikroinstrukcija i koji su funkcija instrukcije koja se izvodi i signala takta. Te funkcije se mogu isčitati direktno iz koda:

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

entity ControlSignalGenerator is
  generic(
    pw_g: integer :=8;    --phi field width
    iw_g: integer :=8;    --instruction input signal width
    cw_g: integer :=13    --cout signal width
  );
  port(
    phi_p: in std_logic_vector(0 to pw_g-1);
    instr_p : in std_logic_vector(iw_g-1 downto 0);
    AC_p : in std_logic;
    reset_p : out std_logic;
    cout_p: out std_logic_vector(0 to cw_g-1) );
  end ControlSignalGenerator;

architecture Behavioral of ControlSignalGenerator is
begin

  process(phi_p,instr_p,AC_p)begin
    cout_p(0) <= phi_p(7)and(instr_p(2));
    cout_p(1) <= phi_p(7)and instr_p(3);
    cout_p(2) <= phi_p(4)and instr_p(6);
    cout_p(3) <= phi_p(1) or phi_p(2) or ( phi_p(5) and (instr_p(0) or instr_p(2)
                  or instr_p(3)) ) or ( phi_p(6) and (instr_p(0) or instr_p(2) or instr_p(3)) );
    cout_p(4) <= (phi_p(6)or phi_p(7) )and instr_p(1);
    cout_p(5) <= phi_p(5)and instr_p(1);
    cout_p(6) <= phi_p(7)and instr_p(0);
    cout_p(7) <= phi_p(4)and(instr_p(0) or instr_p(1) or instr_p(2) or instr_p(3) );
    cout_p(8) <= phi_p(4)and(instr_p(4) or (instr_p(5) and AC_p));
```

```
    cout_p(9) <= phi_p(3);
    cout_p(10) <= phi_p(0);
    cout_p(11) <= phi_p(3);
    cout_p(12) <= phi_p(4)and instr_p(7);
    reset_p <= phi_p(4)and(instr_p(4) or instr_p(5) or instr_p(6) or instr_p(7));
  end process;

end Behavioral;
```

Tako je npr. signal c4 aktivan (pisanje u memoriju) ako se radi o instrukciji st koja je dekodirana kao 01000000 i ako je brojilo odbrojalo 6 ili 7 ciklusa od početka izvođenja instrukcije (jer mikroinstrukcija WRITE traje 2 takta), a to je zapisano kao cout_p <= (phi_p(6) or phi_p (7)) and instr_p(1).

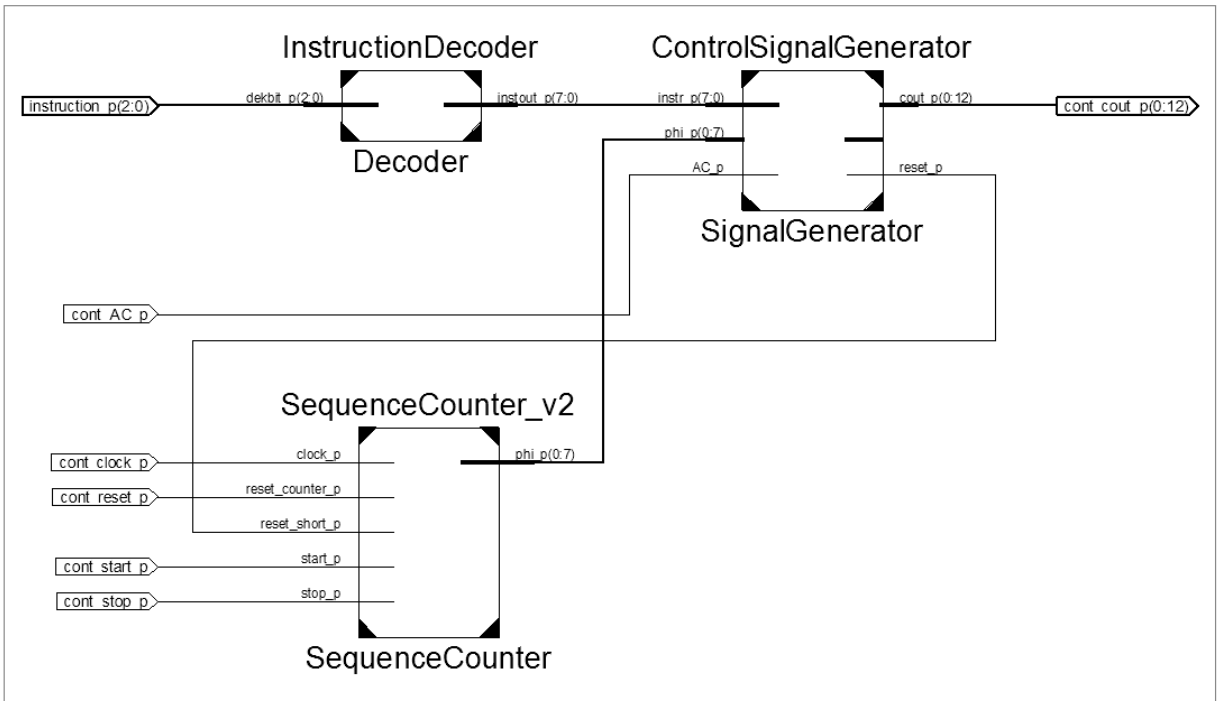
2.1.4 Povezivanje komponenti u cijelinu - upravljačka jedinica

VHDL opis upravljačke jedinice se sastoji od instanciranja komponenti koje ju čine:

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

entity controlUnit is
  generic(
    iw_g: integer :=3;           --širina instrukcije
    cw_g: integer :=13;       --širina izlaza
    dw_g: integer :=8;          --širina dekodirane instrukcije
    pw_g: integer :=8           --širina izlaza brojača sekvenci
  );
  port( instruction_p : in std_logic_vector(iw_g - 1 downto 0);
    cont_clock_p : in std_logic;
    cont_AC_p : in std_logic;
    cont_start_p : in std_logic;
    cont_stop_p : in std_logic;
    cont_reset_p : in std_logic;
    cont_cout_p : out std_logic_vector(0 to cw_g-1) );
  end controlUnit;
```

```
architecture Behavioral of controlUnit is
  signal instr_out_s : std_logic_vector(dw_g-1 downto 0);
  signal phi_s : std_logic_vector(0 to pw_g-1);
  signal reset_short_s : std_logic;
begin

  Decoder:entity work.InstructionDecoder
    generic map(3,8)
    port map(instruction_p,instr_out_s);

  SequenceCounter:entity work.SequenceCounter_v2
    generic map(8)
    port map(cont_reset_p,reset_short_s,cont_start_p,cont_stop_p,cont_clock_p,phi_s);

  SignalGenerator:entity work.controlSignalGenerator
    generic map(8,8,13)
    port map(phi_s,instr_out_s,cont_AC_p,reset_short_s,cont_cout_p);

end Behavioral;
```

Svaka komponenta se instancira generički, s parametrima (npr. dekoder s širinom ulaza 3 i izlaza 4).Sinteza u Xilinx-u daje ovakovu shemu:



2.2 Aritmetičko logička jedinica

Aritmetičko logička jedinica obavlja operaciju zbrajanja dvaju višebitnih podataka, operaciju konjugacije dva signala, te operaciju negacije višebitnog podatka. VHDL implementacija modula koji obavljaju ove operacije je prirodna, i nisu potrebna dodatna pojašnjenja.

2.2.1 Zbrajalo

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity alu_adder is
  generic(
    nd_g: integer := 11            --širina podatkovne sabirnice
  );
  port (
    clock_p : in std_logic;
    inputa_p,inputb_p : in std_logic_vector(nd_g-1 downto 0);
    outputc_p : out std_logic_vector(nd_g-1 downto 0)
  );
end alu_adder;

architecture behavioral of alu_adder is
  signal output_s : std_logic_vector(nd_g-1 downto 0);
begin

  process(clock_p,inputa_p,inputb_p)
    variable a_v : integer;
    variable b_v : integer;
    variable c_v : integer;
  begin
    a_v := conv_integer(inputa_p);
    b_v := conv_integer(inputb_p);
      if rising_edge(clock_p) then
        c_v := a_v + b_v;
        output_s <= conv_std_logic_vector(c_v,nd_g);
      end if;
  end process;

  outputc_p <= output_s;

  end behavioral;
```

2.2.2 Sklop za konjugaciju

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

entity alu_ander is
  generic(
    nd_g: integer := 11            --data bus width
  );
  port (
    clock_p : in std_logic;
    inputa_p,inputb_p : in std_logic_vector(nd_g-1 downto 0);
    outputc_p : out std_logic_vector(nd_g-1 downto 0)
  );
end alu_ander;

architecture behavioral of alu_ander is
  signal output_s : std_logic_vector(nd_g-1 downto 0);
begin

  process(clock_p,inputa_p,inputb_p)begin
    if rising_edge(clock_p)then
      for i in 0 to nd_g-1 loop
        output_s(i) <= inputa_p(i)and inputb_p(i);
      end loop;
    end if;
  end process;

  outputc_p <= output_s;

end behavioral;
```

2.2.3 Sklop za bitovnu negaciju

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

entity alu_noter is
  generic(
    nd_g: integer := 11            --data bus width
   );
  port (
    clock_p : in std_logic;
    inputa_p : in std_logic_vector(nd_g-1 downto 0);
    outputc_p : out std_logic_vector(nd_g-1 downto 0)
  );
end alu_noter;

architecture behavioral of alu_noter is
  signal output_s : std_logic_vector(nd_g-1 downto 0);
begin

  process(clock_p,inputa_p) begin
    if rising_edge(clock_p) then
      for i in 0 to nd_g-1 loop
        output_s(i) <= not inputa_p(i);
      end loop;
    end if;
  end process;

outputc_p <= output_s;

end behavioral;
```

2.2.4 Povezivanje komponenti u cijelinu – ALU jedinica

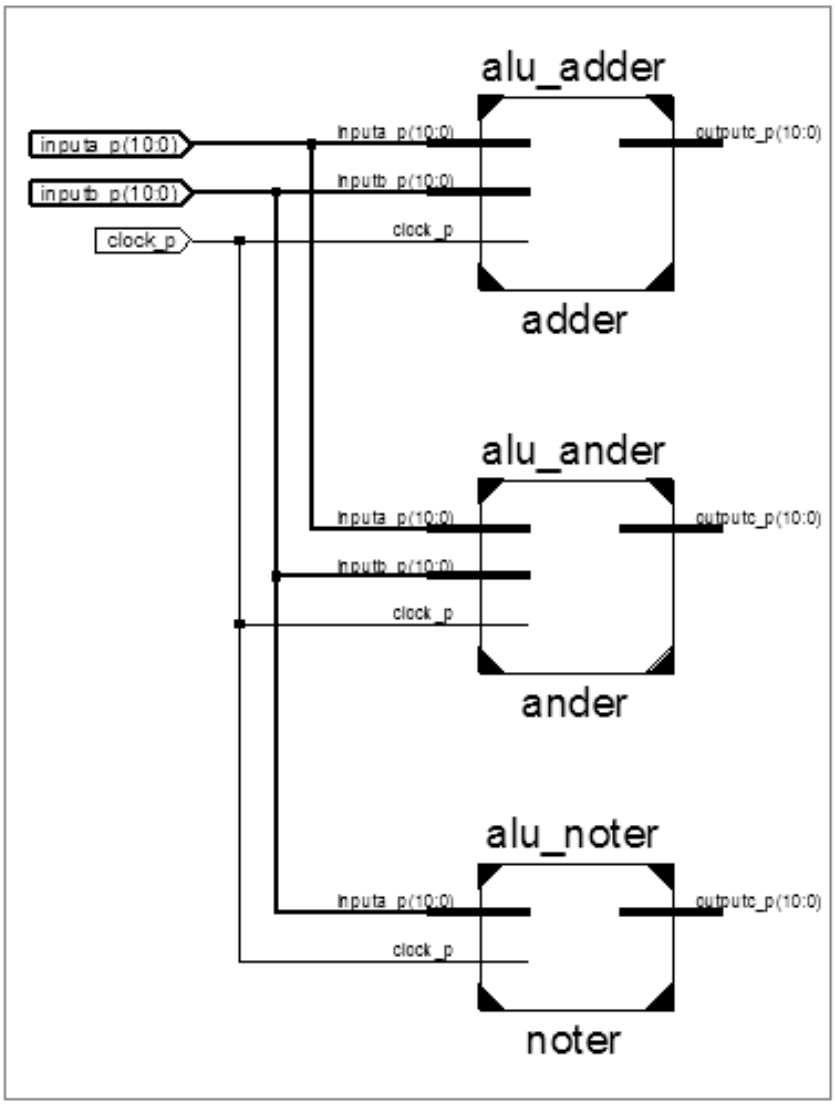
Slično kao i kod upravljačke jedinice, ALU jedinica instancira navedene module. Dodatno, ALU mora, ovisno o operaciji, odrediti koji rezultat prosljeđuje na svoje izlaze – rezultat operacije (add, and ili not). Stoga njen VHDL kod sadrži dodatni proces koji upravlja izlazom. Također, postoji i proces koji postavlja ZERO zastavicu: ako je rezultat operacije 0 onda se zastavica postavlja u 1 inače u 0.

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

entity AL_Unit is
  generic(
    nd_g: integer := 11           --širina podatkovne sabirnice
  );
  port (
    clock_p : in std_logic;
    c_operation_p : in std_logic_vector(0 to 2);
    c_writeAC_p : out std_logic;--signalizira da je rezultat spreman,
                                --te se može upisati u akumulator
    ac_zero_p : out std_logic;
    inputa_p : in std_logic_vector(nd_g-1 downto 0);
    inputb_p : in std_logic_vector(nd_g-1 downto 0);
    outputc_p : out std_logic_vector(nd_g-1 downto 0)
  );
end AL_Unit;

architecture Behavioral of AL_Unit is
  signal outadder_s : std_logic_vector(nd_g-1 downto 0);
  signal outander_s : std_logic_vector(nd_g-1 downto 0);
  signal outnoter_s : std_logic_vector(nd_g-1 downto 0);
  signal aluout_s : std_logic_vector(nd_g-1 downto 0);
begin

  adder:entity work.alu_adder
    generic map(nd_g)
    port map(clock_p,inputa_p,inputb_p,outadder_s);
```

```
ander:entity work.alu_ander
    generic map(nd_g)
    port map(clock_p,inputa_p,inputb_p,outander_s);

  noter:entity work.alu_noter
    generic map(nd_g)
    port map(clock_p,inputb_p,outnoter_s);
  --ovisno o operaciji, izabire se rezultat koji se postavlja na
  --izlaznu sabirnicu ALU jedinice
  output:process (c_operation_p,outadder_s,outander_s,outnoter_s) begin
    case c_operation_p is
      when "100" => aluout_s <= outadder_s; c_writeAC_p <= '1';
      when "010" => aluout_s <= outander_s; c_writeAC_p <= '1';
      when "001" => aluout_s <= outnoter_s; c_writeAC_p <= '1';
      when others => null; c_writeAC_p <= '0';
    end case;
  end process;
  --postavljanje ZERO zastavice
  ac_flag:process(aluout_s)begin
    if aluout_s = ( aluout_s'range => '0' ) then
      ac_zero_p <= '1';
    else
      ac_zero_p <= '0';
    end if;
  end process;

outputc_p <= aluout_s;

end Behavioral;
```

Može se uočiti da je ALU jedinica za razliku od upravljačke potpuno generička: parametar pri njenom instanciranju je širina podatkovne sabirnice, odnosno ulaznog podatka, i pri promjeni tog parametra nije potrebno unositi dodatne izmjene u kod. Isto vrijedi i za module zbrajanja, logičkog AND i NOT, koji su također potpuno generički.

Sinteza u Xilinx-u prepoznaje osnovne elemente ALU jedinice (pa tako modul za zbrajanje prepoznaje kao 8-bitno zbrajalo), a shema je na slijedećoj slici:



Na shemi, zbog jasnoće, nisu prikazani multipleksor i ostali signali.

2.3 Skup registara

Skup registara procesora obuhvaća adresni (MAR), podatkovni (MDR) i instrukcijski (IR), te programsko brojilo (PC) i akumulator (AC). Osim akumulatora, svi registiri imaju istu strukturu, samo su drukčije njihove širine. Ulaz svakog registra čine dvije sabirnice, i dva upravljačka bita. Ovisno o vrijednosti tih bitova, u registar se upisuje vrijednost s prve ili druge sabirnice. Tako npr. ulaz u MAR čine podatci iz PC i MDR registra. Ovisno o vrijednosti upravljačkih bitova c7 i c10 koji su spojeni na ulaz registra, upisuje se podatak s MDR ili PC sabirnice. Zbog iste strukture,registri se grade instanciranjem jedne komponente, u kodu nazvane CPU_Register. Akumulator ima dodatnu funkciju posmaka u desno (radi ostvarenja instrukcije shr) , stoga se on implementira posebno (iako nema puno razlike s obzirom na ostale registre).

2.3.1 ZERO zastavica

Kao što je već rečeno, ova zastavica služi pohranjivanju rezultata testiranja: je li vrijednost u akumulatoru jednaka 0 ili ne. VHDL implementacija je maksimalno jednostavna, i predstavlja asinkroni bistabil, a navodi se zbog kompletnosti:

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

entity ACZeroFlag is
  port(
    set_p : in std_logic;
    reset_p : in std_logic;
    InAc_P : in std_logic;
    OutAc_p : out std_logic
  );
end ACZeroFlag;

architecture Behavioral of ACZeroFlag is
  signal out_s : std_logic;
begin

  process (InAc_P,set_p,reset_p) begin
    if reset_p = '1' then
```

```
        out_s <= '0';
      elsif set_p = '1' then
        out_s <= InAc_p;
      end if;
    end process;

    OutAc_p <= out_s;

  end Behavioral;
```

2.3.2 *Komponenta CPU_Register*

Komponenta služi za općenito ostvarenje registra procesora. Ima dvije ulazne sabirnice i dva upravljačka signala (sel0 i sel1) na temelju kojih se odabire podatak koji se upisuje u registar. VHDL opis dan u nastavku:

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

entity CPU_Register is
  generic(
    n_g: integer := 8
  );
  port (
    reset_p : in std_logic;
    sel0_p : in std_logic;
    sel1_p : in std_logic;
    inArrayA_p : in std_logic_vector(n_g-1 downto 0);
    inArrayB_p : in std_logic_vector(n_g-1 downto 0);
    outArray_p : out std_logic_vector(n_g-1 downto 0)
  );
end CPU_Register;

architecture Behavioral of CPU_Register is
  signal outArray_s : std_logic_vector(n_g-1 downto 0);
begin

  MUX:process(inArrayA_p,inArrayB_p,sel0_p,sel1_p,reset_p) begin
    if reset_p = '1' then outArray_s <= (others =>'0');
    elsif sel0_p = '1' then outArray_s <= inArrayA_p;
    elsif sel1_p = '1' then outArray_s <= inArrayB_p;
    else null;
    end if;
  end process;

  outArray_p <= outArray_s;

end Behavioral;
```

2.3.3 Akumulator

Razlika „običnog“ registra i akumulatora je kao što je već navedeno mogućnost pomaka u desno (shift registar). Sve ostalo je isto, osim što ovdje postoje tri upravljačka signala (dva za odabir ulaznog signala i treći koji govori da se podatak u akumulatoru mora pomaći udesno).

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

entity acumulator is
  generic(
    n_g: integer := 8
  );
  port (
    clock_p : in std_logic;
    c_12_p : in std_logic;     --shr control
    c_6_p : in std_logic;      --write control
    c_alu_p : in std_logic;    --alu write control
    inArray_fromMDR_p : in std_logic_vector(n_g-1 downto 0);
    inArray_fromALU_p : in std_logic_vector(n_g-1 downto 0);
    outArray_p : out std_logic_vector(n_g-1 downto 0)
  );
end acumulator;

architecture Behavioral of acumulator is
  signal outArray_s : std_logic_vector(0 to n_g-1);
begin

  process(clock_p,c_alu_p,c_12_p,c_6_p,inArray_fromALU_p,inArray_fromMDR_p) begin
    if rising_edge(clock_p) then
      if c_alu_p = '1' then outArray_s <= inArray_fromALU_p;
      elsif c_6_p = '1' then outArray_s <= inArray_fromMDR_p;
      elsif c_12_p = '1' then outArray_s <= outArray_s(n_g-1) & outArray_s(0 to n_g-1-1);
      else null;
      end if;
    end if;
  end process;

outArray_p <= outArray_s;

end Behavioral;
```

2.3.4 PC inkrementer

Kako bi se podržala mikroinsrukcija uvećavanja programskog brojila za 1, implementirana je komponenta PC_incrementer. Mada se ona mogla izvesti kao instanca ALU jedinke odnosno sklopa za zbrajanje, ili se to zrajanje moglo obaviti u samoj ALU (ipak, u tom slučaju bi ona morala trajati duže) , najbolje rješenje je da se ona implementira posebnim kodom. Naime, zbrajanje PC + 1 je jednostavnije od običnog. Zadnja znamenka rezultata se dobije kao R7 = A7 + 1, gdje je A7 zadnja znamenka ulaza. Ovo je ekvivalento operaciji NOT nad A7, te je R7 = NOT A7. Prijenos na više mjesto se događa ako je A7 = 1, pa je stoga C7 (prijenos s 7 mjesta) jednak C7 = A7. Pretposljednja znamenka rezultata se onda dobiva kao R6 = A6 XOR C7 = A6 XOR A7, a C6 = A6 AND C7 = A6 AND A7, budući i A6 i C7 moraju biti 1 da bi se dogodio prijenos na više mjesto. Na isti način je R5 = A5 XOR C6 = A5 XOR (A6 AND A7), C6 = A5 AND C6 = A5 AND A6 AND A7, itd. Iz posljednje formule vidimo da ćemo dobiti zbrajalo s propagacijskim prenosom. Dakle, općenito bi bilo :

Rn = NOT An; Rk = Ak XOR (Ak+1 AND ... AND An)

U VHDL-u se to ostvaruje pomoću dvije for petlje :

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

entity pc_incrementer is
  generic(
    n_g: integer := 8
  );
  port (
    c_increment_p : in std_logic;     --increment control
    inArray_p : in std_logic_vector(n_g-1 downto 0);
    outArray_p : out std_logic_vector(n_g-1 downto 0)
  );
end pc_incrementer;

architecture Behavioral of pc_incrementer is
  signal outArray_s : std_logic_vector(n_g-1 downto 0);
begin
```

```
  process(c_increment_p,inArray_p)
    variable tmp : std_logic;
  begin
    if rising_edge(c_increment_p) then
      outArray_s(0) <= not inArray_p(0);
        for i in 1 to n_g-1 loop
          tmp := '1';
            for j in 0 to i-1 loop
              tmp := tmp and inArray_p(j);
            end loop;
          outArray_s(i) <= tmp xor inArray_p(i);
        end loop;
    else null;
    end if;
  end process;

  outArray_p <= outArray_s;

end Behavioral;
```

2.4 Povezivanje komponenti u cjelinu - procesor

Nakon što su napravljene sve komponente procesora, one se spajaju u cjelinu, prema shemi. Postoji i mala razlika: ovaj procesor ima dvije podatkovne sabirnice – dolaznu i odlaznu, zbog jednostavnije izvedbe. Ulaz u procesor je signal takta, te signali za pokretanje/resetiranje/zaustavljanje, te podatkovna sabirnica sa podatcima iz memorije. Izlaz su upravljački signali za čitanje i pisanje memorije, adresna sabirnica i podatkovna sabirnica prema memoriji.

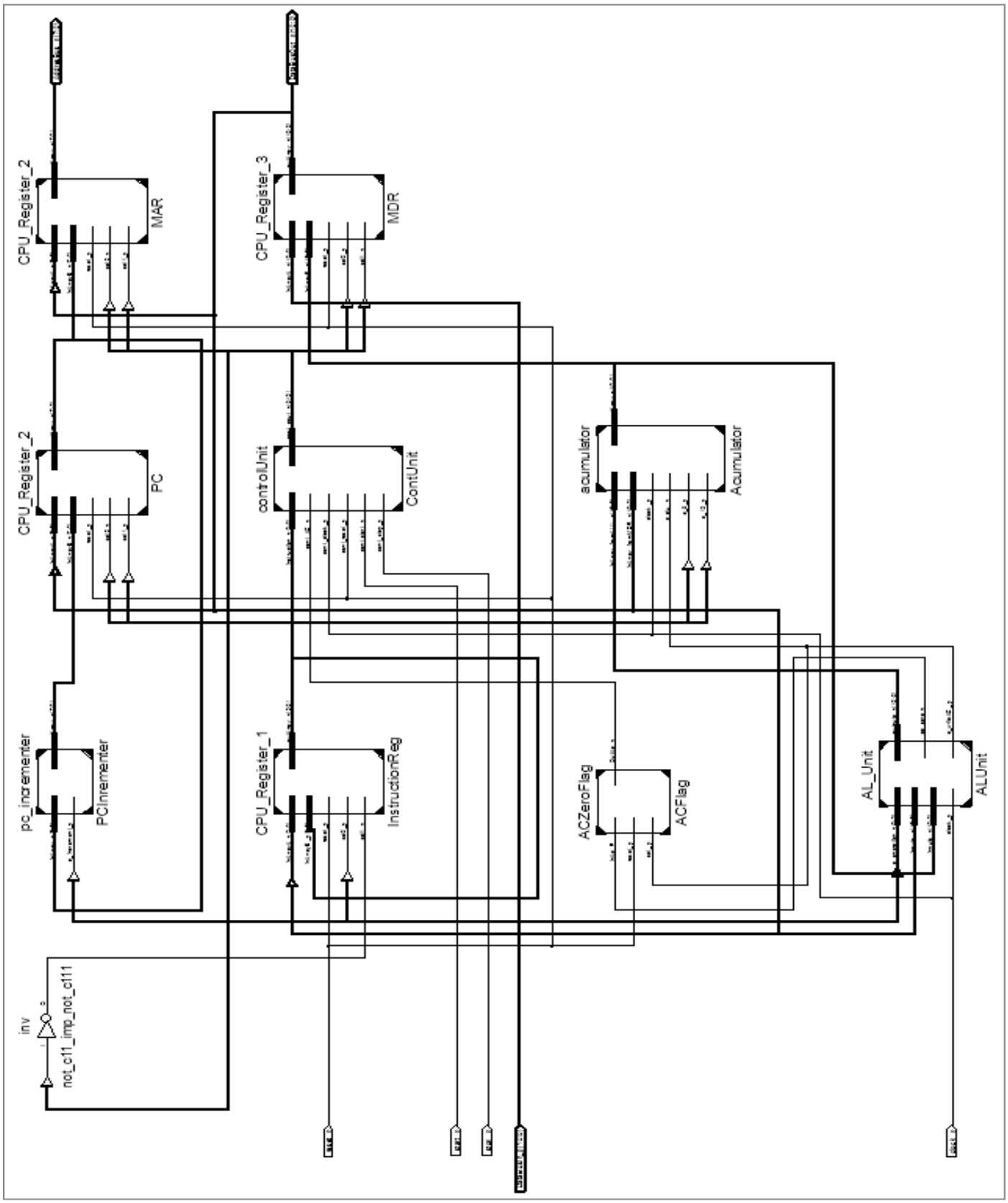
Slijedi VHDL kod procesora :

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

entity Simple_CPU is
  generic(
    nd_g: integer := 11;          --širina podatkovne sabirnice
    na_g: integer := 8;              --širina adresne sabirnice
    iw_g: integer := 3;           --širina instrukcije
    cw_g: integer :=13;         --širina kontrolne sabirnice
    dw_g: integer :=8;            --širina dekodirane instrukcije
    pw_g: integer :=8             --širina sabirnice brojača sekvence
    );
    port(
    clock_p : in std_logic;
    reset_p : in std_logic;
    start_p : in std_logic;
    stop_p : in std_logic;
    DataBusIn_p : in std_logic_vector(nd_g-1 downto 0);
    memRead_p : out std_logic;
    memWrite_p : out std_logic;
    DataBusOut_p : out std_logic_vector(nd_g-1 downto 0);
    AddresBus_p : out std_logic_vector(na_g-1 downto 0)
    );
end Simple_CPU;
```

```
architecture Behavioral of Simple_CPU is
  signal ACI_s : std_logic;
  signal ACO_s : std_logic;
  signal CB_s : std_logic_vector(0 to cw_g-1);
  signal MDRBus_s : std_logic_vector(nd_g-1 downto 0);
  signal IROut_s : std_logic_vector(0 to iw_g-1);
  signal c_writeAC_s : std_logic;
  signal ALU_out_s : std_logic_vector(nd_g-1 downto 0);
  signal ACOut_s : std_logic_vector(nd_g-1 downto 0);
  signal PCOut_s : std_logic_vector(na_g-1 downto 0);
  signal PCIncOut_s : std_logic_vector(na_g-1 downto 0);
  signal not_c11 : std_logic;
begin

  ACFlag:entity work.ACZeroFlag
    port map(c_writeAC_s,reset_p,ACI_s,ACO_s);

  ALUnit:entity work.AL_Unit
    generic map(nd_g)
    port map(clock_p, c_operation_p(0 to 2)=>CB_s(0 to 2),
             c_writeAC_p=>c_writeAC_s,
             ac_zero_p=>ACI_s,
             inputa_p=>MDRBus_s,
             inputb_p=>ACOut_s,
             outputc_p=>ALU_out_s);

  Acumulator:entity work.acumulator
    generic map(nd_g)
    port map(clock_p,CB_s(12),CB_s(6),c_writeAC_s,MDRBus_s,ALU_out_s,ACOut_s);

  not_c11 <= not CB_s(11);
  -- budući instrukcijski registar ima samo jedan ulaz, na drugi ulaz je spojen
  -- izlaz iz samog instrukcijskog registra
 InstructionReg:entity work.CPU_Register
   generic map(iw_g)
   port map(reset_p,CB_s(11),not_c11,
            inArrayA_p(2 downto 0)=>MDRBus_s(10 downto 8),
            inArrayB_p=>IROut_s,
            outArray_p=>IROut_s);
```

```
ContUnit:entity work.ControlUnit
  generic map(iw_g,cw_g,dw_g,pw_g)
  port map(IROut_s,clock_p,ACO_s,start_p,stop_p,reset_p,CB_s);


PCInrementer:entity work.pc_incrementer
  generic map(na_g)
  port map(CB_s(9),PCOut_s,PCIncOut_s);

PC:entity work.CPU_Register
  generic map(na_g)
  port map(reset_p,CB_s(8),CB_s(9),
           inArrayA_p(7 downto 0)=>MDRBus_s(7 downto 0),
           inArrayB_p=>PCIncOut_s,
           outArray_p=>PCOut_s);

MAR:entity work.CPU_Register
  generic map(na_g)
  port map(reset_p,CB_s(7),CB_s(10),
  inArrayA_p(7 downto 0)=>MDRBus_s(7 downto 0),
  inArrayB_p=>PCOut_s,
  outArray_p=>AddresBus_p);

MDR:entity work.CPU_Register
  generic map(nd_g)
  port map(reset_p,CB_s(3),CB_s(5),
           DataBusIn_p,
           ACOut_s,
           MDRBus_s);

DataBusOut_p <= MDRBus_s;
memRead_p <= CB_s(3);
memWrite_p <= CB_s(4);

end Behavioral;
```

Prospajanje se vrši prema shemi i nije ga potrebno dodatno objašnjavati. Na sljedećoj je slici shema koju stvara sinteza u Xilinx – u:

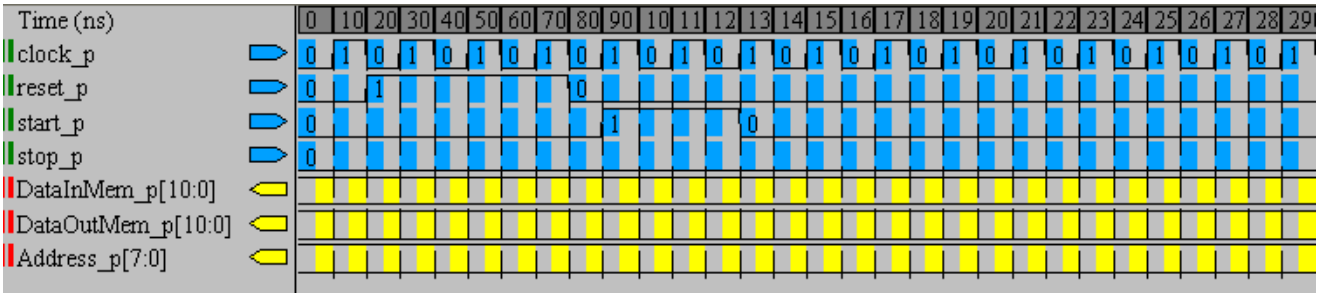

CPU_Register_2
MAR
CPU_Register_3
MDR
CPU_Register_2
PC
controlUnit
ContUnit
acumulator
Acumulator
pc_inrementer
PCInrementer
CPU_Register_1
InstructionReg
ACZeroFlag
ACFlag
AL_Unit
ALUnit
inv
not_c11_imp_not_c111

3. Simulacija

Simulacija je zapravo testiranje sklopa – na ulaz se dovedu testni podatci (signali) i zatim se promatra izlaz. Ako je on u skladu s očekivanim rezultatima, sklop radi ispravno. Simulaciju je potrebno provesti za svaki sklop u sustavu, stoga je i u ovom projektu svaki sklop simuliran prije nego je bio upotrebljen u gradnji sustava. Ovdje će se navesti samo simulacija cijelog procesora.

Simulacija se radi tako da se za odabranu komponentu napravi testna okolina. Testna se pak okolina može izraditi u VHDL-u ili pak pomoću modula Test Wave Bench u Xilinx-u (ipak, ovaj je alat izbačen u verziji 10 Xilinx alata). Drugi pristup je nešto brži, međutim pisanje vlastite testne okoline ima puno više prednosti. Zbog nedostatka vremena, u ovom projektu simulacija je napravljena pomoću Test Wave Bench-a, u sklopu razvojne okoline Xilinx Web ISE 7.

Testni je modul prikazan na slici:

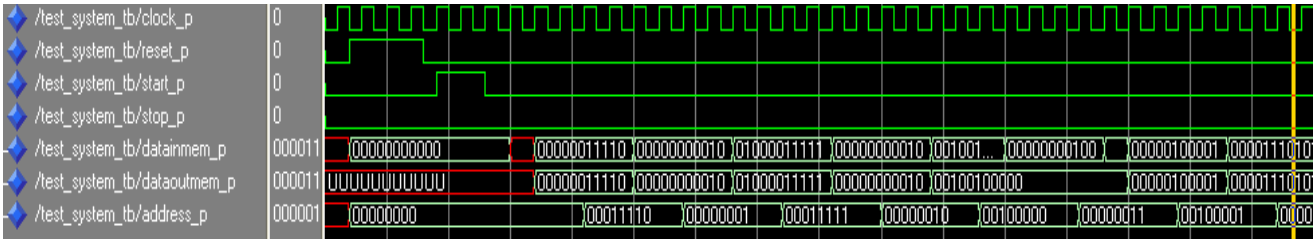


Ulazni signali modula za testiranje su signal takta (clock_p) i signali za upravljanje (reset_p, start_p, stop_p), a izlazni podatkovne i adresna sabirnica. Na ovaj način možemo vidjeti kakvo je stanje na tim sabirnicama u procesoru za vrijeme izvođenja instrukcija u memoriji. Na procesor je spojena memorija čiji je sadržaj (prve tri lokacije):

```
"00000011110",  --ld 30
"01000011111",  --add 31
"00100100000",  --st 32
```

Također, vidimo da je testni scenarij takav, da se prvo procesor resetira (reset_p je visoko), a zatim pokrene (start_p je visoko). Dakle, procesor se pokreće, a kao rezultat, očekujemo da je na 32 memorijskoj lokaciji zbroj vrijednosti koje su upisane na memorijskim lokacijama 30 i 31.

Simulator korišten u ovom projektu je ModelSim 6.5 SE, a u Xilinx Web ISE 7 se pokreće klikom na Simulate Behavioral Model. Rezultat simulacije može se vidjeti na slijedećoj slici:



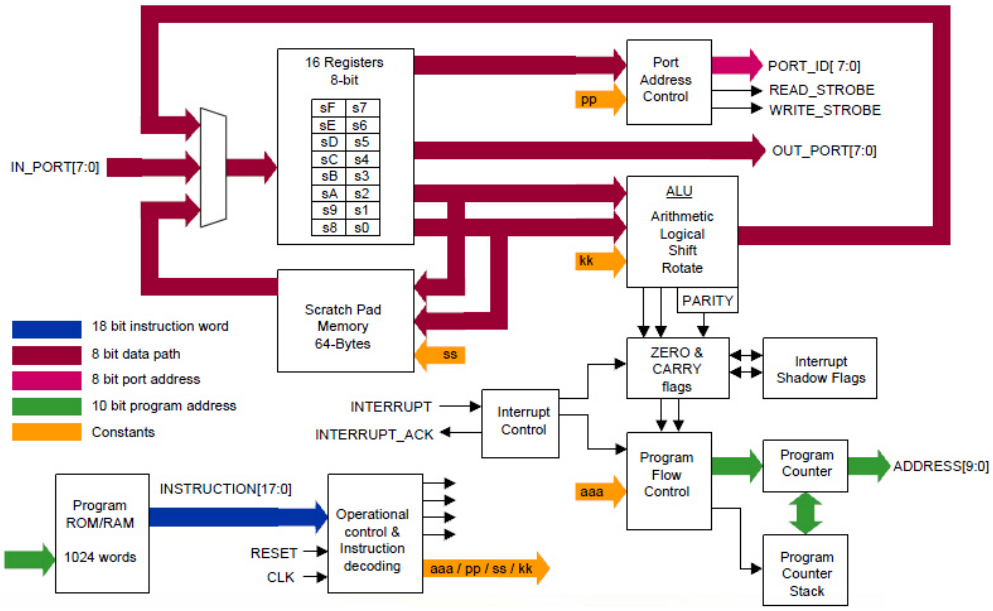
Možemo dakle isčitati stanje na sabirnicama procesora. Tako npr. vidimo da se na adresnoj sabirnici izmjenjuju adrese instrukcija (0, 1 i 2 za naš odsječak) i adrese operanada (30, 31 i 32). Na podatkovnim se pak izmjenjuju instrukcijski kodovi i vrijednosti operanada.

4. Sinteza na stvarnoj pločici – Spartan 3E

Dosad se procesor razmatrao teoretski – pomoću shema i VHDL opisa. Također, u alatu Xilinx provedena je i sinteza, da bi se provjerilo je li uopće moguće dani opis sklopovlja pomoću VHDL opisa pretočiti u stvaran sustav. Nakon uspješno provedenih simulacija i testiranja pristupa se implementaciji na stvarnoj platformi, koja je u ovom slučaju pločica Spartan 3E. Međutim, u ovom koraku nastaju problemi: naš procesor nema nikakvih ulazno-izlaznih jedinica kojim bi ga povezali sa stvarnim svijetom. Kao moguća rješenja nameću se izmjena arhitekture ili pomoć nekih drugih sklopova (procesora). Prvo rješenje nije najsretnije, jer tada to nebi više bio originalni procesor koji se želio implementirati. Drugo rješenje je bolje, jer se ne moraju raditi nikakve izmjene. Za „pomoćni“ je procesor izabran picoBlaze (odnosno KCPSM3), soft procesor koji se nalazi u Xilinx paketu.

4.1 PicoBlaze

PicoBlaze je soft mikroprocesor, isprogramiran u VHDL-u, potpuno otvorenog koda. Širina sabirnica mu je 8 – bitna, a može se programirati u assembleru. Također, posjeduje i ulazno-izlazne instrukcije koje omogućuju spajanje i komunikaciju s vanjskim uređajima. Ima jednostavnu arhitekturu :



Za ovaj procesor također postoji grafički program – pBlazeIDE, u kojem se picoBlaze može programirati sa svojim skupom instrukcija, a zatim i simulirati. To uvelike olakšava rad s njim. Nakon što je program napisan, on se sprema u vhdl datoteku u kojoj je zapravo definicija programske memorije (ProgramROM) koja se spaja na picoBlaze. Spajanje RAM memorije na picoBlaze se vrši preko izlaznih sabirnica out_port i port_id. Upravo će ta memorija biti ono što će povezivati dva procesora – oba će biti spojena na nju.

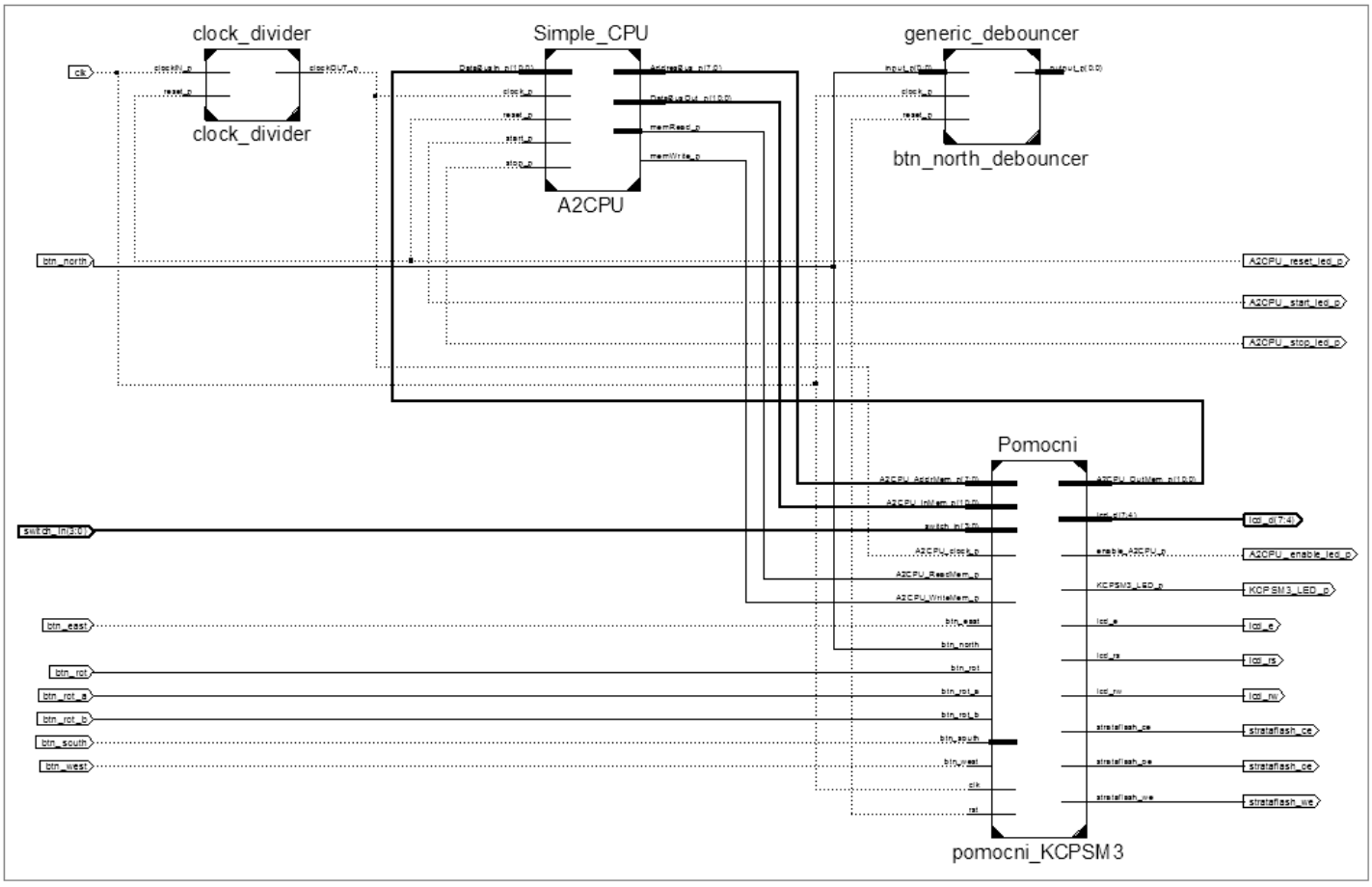
4.2 Ulazno – izlazni sklopovi na Spartan 3E pločici

Pogledajmo sada konkretno na što se odnose ulazno izlazni sklopovi na pločici:

Crveno zaokruženo su četiri slide switcha koji su iskorišteni za definiranje podataka : svaki može biti u jednom od dva stanja, pa se tako može iskombinirati bilo koja znamenka 0-F(hex). Žuto su LED diode koje mogu poslužiti kao indikatori. Plavo je LCD ekran, a smeđe su pet tipkala, s tim da se gumb u sredini može dodatno i rotirati u lijevu ili desnu stranu. Pritisak na te gumbe generira prekid na picoBlaze procesoru, koji odlazi u rutinu za obradu prekida, u kojoj se obavlja glavni dio posla.

4.3 Povezivanje dvaju procesora

Kao što je rečeno, dva procesora će dijeliti memoriju – picoBlaze će upisivati u memoriju stanje ulaznih jedinica i ispisivati njeno stanje na LCD. Naš procesor (nazvat ćemo ga A2CPU) će čitati podatke iz memorije i u nju pisati. Shema sustava dana je slikom :



Zajednička se memorija nalazi unutar sklopa pomocni_KCPSM3, gdje je smješten i picoBlaze. A2CPU joj pristupa preko sabirnica A2CPU_InMem_p, A2CPU_OutMem_p , A2CPU_AddrMem_p te upravljačkih signala. Čitav se sklop sa stvarnim signalima povezuje pomoću ucf datoteke.

4.3.1 Programski opis

Sama izvedba sklopa pomocni_KCPSM3 nije u fokusu ovog projekta, pa neće biti detaljno razmatrana. Ovjde se samo navodi kod. Pritom VHDL kod služi za spajanje picoBlaze procesora s njegovom programskom memorijom, vanjskom RAM memorijom te ulazno-izlaznim uređajima.

4.3.1.1 VHDL kod sklopa pomocni_KCPSM3

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

entity Pomocni is
  port (
    strataflash_oe : out std_logic;
    strataflash_ce : out std_logic;
    strataflash_we : out std_logic;
    switch_in : in std_logic_vector(3 downto 0);
    btn_west : in std_logic;
    btn_south : in std_logic;
    btn_north : in std_logic;
    btn_east : in std_logic;
    btn_rot : in std_logic;
    btn_rot_a : in std_logic;
    btn_rot_b : in std_logic;
    KCPSM3_LED_p : out std_logic;
    lcd_d : out std_logic_vector(7 downto 4);
    lcd_rs : out std_logic;
    lcd_rw : out std_logic;
    lcd_e : out std_logic;
    clk : in std_logic;
    rst : in std_logic;
    enable_A2CPU_p : out std_logic;
    A2CPU_clock_p : in std_logic;
    A2CPU_InMem_p : in std_logic_vector(10 downto 0);
    A2CPU_OutMem_p : out std_logic_vector(10 downto 0);
    A2CPU_AddrMem_p : in std_logic_vector(7 downto 0);
    A2CPU_WriteMem_p : in std_logic;
    A2CPU_ReadMem_p : in std_logic
  );
end Pomocni;
```

```
architecture Behavioral of Pomocni is

  component kcpsm3
    port (
      address : out std_logic_vector(9 downto 0);
      instruction : in std_logic_vector(17 downto 0);
      port_id : out std_logic_vector(7 downto 0);
      write_strobe : out std_logic;
      out_port : out std_logic_vector(7 downto 0);
      read_strobe : out std_logic;
      in_port : in std_logic_vector(7 downto 0);
      interrupt : in std_logic;
      interrupt_ack : out std_logic;
      reset : in std_logic;
      clk : in std_logic
      );
  end component;

  signal address : std_logic_vector(9 downto 0);
  signal instruction : std_logic_vector(17 downto (
  signal port_id : std_logic_vector(7 downto 0);  ;
  signal out_port : std_logic_vector(7 downto 0);
  signal in_port : std_logic_vector(7 downto 0);
  signal write_strobe : std_logic;
  signal read_strobe : std_logic;
  signal interrupt : std_logic;
  signal interrupt_ack : std_logic;
  signal Cpu8BitMemAddr_s : std_logic_vector(7 downto 0);
  signal Cpu8BitMemDataIn_s : std_logic_vector(10 downto 0);
  signal Cpu8BitMemDataOut_s : std_logic_vector(10 downto 0);
  signal MemClock_s : std_logic;
  signal SwitchData_s : std_logic_vector(7 downto 0);
  signal ButtonData_s : std_logic_vector(7 downto 0);
  signal LCDData_s : std_logic_vector(7 downto 0);

  signal LCDControlCe : std_logic;
  signal Cpu8BitMem_AddrCe : std_logic;
  signal Cpu8BitMem_Data7to0Ce : std_logic;
  signal Cpu8BitMem_Data10to8Ce : std_logic;
  signal ReadMem_s : std_logic;
  signal WriteMem_s : std_logic;
  signal LedControl : std_logic;
  signal KCPSM_enable_s : std_logic := '1';
  signal KCPSM3_LED_s : std_logic :='1';
begin
```

```
strataflash_oe <= '1';
strataflash_ce <= '1';
strataflash_we <= '1';

processor:kcpsm3
  port map(address,instruction,port_id,write_strobe,out_port,read_strobe,
           in_port,interrupt,interrupt_ack,rst,clk);

program_rom:entity work.ProgRom
  port map(address,instruction,clk);

MemClock_s <= clk when (KCPSM_enable_s = '1') else A2CPU_clock_p ;

Cpu8Bit Mem:entity work.Cpu8BitMem
   port map(MemClock_s,Cpu8BitMemDataIn_s,Cpu8BitMemDataOut_s,Cpu8BitMemAddr_s,
            ReadMem_s,WriteMem_s);

 LCD_Control:entity work.LCDControl
   port map(clk => clk, controlIn => LCDData_s(3 downto 0),lcd_rs => lcd_rs,
            lcd_rw => lcd_rw,lcd_re => lcd_e,dataIn => LCDData_s(7 downto 4),
            dataOut => lcd_d);

 Switch_Control:entity work.SwitchControl
   port map(switch_in,SwitchData_s);

 Button_Control:entity work.ButtonControl
   port map(clk => clk,ButtonIn(0) => btn_west,ButtonIn(1) => btn_south,
            ButtonIn(2) => btn_north,ButtonIn(3) => btn_east,ButtonIn(4) => btn_rot,
            ButtonIn(5) => btn_rot_a,ButtonIn(6) => btn_rot_b,ButtonOut => ButtonData_s);

 interrupt_control: process(clk)begin
   if rising_edge(clk) then
     if interrupt_ack='1' then
       interrupt <= '0';
     else
        interrupt <= ( (btn_west or btn_south or btn_north or btn_east
                        or btn_rot or ButtonData_s(5)) and KCPSM_enable_s
                      ) or ((ButtonData_s(5)) and not KCPSM_enable_s);
     end if;
   end if;
 end process interrupt_control;
```

```
input_ports:process(clk)begin
  if rising_edge(clk) then
    if KCPSM_enable_s = '1' then
      case port_id is
        when "00000100" => in_port <= Cpu8BitMemDataOut_s(7 downto 0);
        when "00000101" => in_port <= "00000"&Cpu8BitMemDataOut_s(10 downto 8);
        when "00001000" => in_port <= SwitchData_s;
        when "00010000" => in_port <= ButtonData_s;
        when others => in_port <= "XXXXXXXX";
      end case;
    else
      A2CPU_OutMem_p <= Cpu8BitMemDataOut_s;
        if port_id = "00010000" then
            in_port <= "00"&ButtonData_s(5)&"00000";
          else
            in_port <= "XXXXXXXX";
          end if;
    end if;
  end if;
end process;

output_ports: process (clk)begin
  if rising_edge(clk) then

    if LedControl = '1' then
       KCPSM3_LED_p <= out_port(0);
        KCPSM_enable_s <= out_port(0);
        enable_A2CPU_p <= not out_port(0);
     end if;

    if KCPSM_enable_s = '1' then
      if LCDControlCe = '1' then
          LCDData_s <= out_port;
        end if;
      if Cpu8BitMem_AddrCe = '1' then
          Cpu8BitMemAddr_s <= out_port;
        end if;
      if Cpu8BitMem_Data7to0Ce = '1' then
          Cpu8BitMemDataIn_s(7 downto 0) <= out_port;
        end if;
      if Cpu8BitMem_Data10to8Ce = '1' then
          Cpu8BitMemDataIn s(10 downto 8) <= out_port(2 downto 0);
      end if;
  else
    Cpu8BitMemDataIn_s <= A2CPU_InMem_p;
    Cpu8BitMemAddr_s <= A2CPU_AddrMem_p;
  end if;

  end if;
end process;
```

```
    LCDControlCe <= '1' when (write_strobe = '1' and port_id = "00000001") else '0';
    Cpu8BitMem_AddrCe <= '1' when (write_strobe = '1' and port_id = "00000010") else '0';
    ReadMem_s <= '1' when ((write_strobe = '1' and port_id = "01000000" and KCPSM_enable_s = '1'
                          ) or (A2CPU_ReadMem_p ='1' and KCPSM_enable_s = '0')) else '0';
    WriteMem_s <= '1' when ((write_strobe = '1' and port_id = "10000000" and KCPSM_enable_s = '1'
                           ) or (A2CPU_WriteMem_p = '1' and  KCPSM_enable_s = '0')) else '0';
    Cpu8BitMem_Data7to0Ce <= '1' when (write_strobe = '1' and port_id = "00000100") else '0';
    Cpu8BitMem_Data10to8Ce <= '1' when (write_strobe = '1' and port_id = "00000101") else '0';
    LedControl <= '1' when (write_strobe = '1' and port_id = "00100000")else '0';

end Behavioral;
```

Gornji kod instancira potrebne komponente (picoBlaze i njegovu programsku memoriju, kontrolere vanjskih jedinica te djeljenu memoriju) i implementira sklopove za povezivanje: procesi interupt_control, input i output ports, te kombinacijski skolopovi na kraju koda.

4.3.1.2 Kod prekidnog potprograma PicoBlaze-a

PicoBlaze kod u assembleru sadrži preko 600 linija i nema smisla ga cijelog navoditi. U sljedećem se listingu navodi samo prekidna rutina koja obavlja većinu posla (ostatak se odnosi na upravljanje LCD – om i ispisivanjem na isti):

```
; ******************************************************************************
; Prekidna rutina
; ******************************************************************************

ISR:
otkrij_uzrok:
                    IN      stanje_gumba, button_port
                    IN      stanje_prekidaca, switch_port
provjeri_btn_west:  COMP    stanje_gumba, 1             ;upis hex znamenke
                    JUMP    NZ, provjeri_btn_south
                    LOAD    sF, stanje_prekidaca
                    FETCH   stanje_moda_upisa, mod_upisa
                    COMP    stanje_moda_upisa, 1
                    JUMP    NZ, mod_ucitaj_data
                    FETCH   stanje_brojaca, brojac_znamenki
```

```
mod_ucitaj_addr:    COMP    stanje_brojaca, 1
                    JUMP    NZ, donji_nibl
                    CALL    LCD_Reset
                    CALL    disp_Addr_por
gornji_nibl:        SL0     stanje_prekidaca
                    SL0     stanje_prekidaca
                    SL0     stanje_prekidaca
                    SL0     stanje_prekidaca
                    STORE   stanje_prekidaca, ocitana_lokacija
                    LOAD    s0, sF
                    CALL    ispisi_s0
                    JUMP    smanji_brojac
donji_nibl:         FETCH   s0, ocitana_lokacija
                    OR      stanje_prekidaca, s0
                    STORE   stanje_prekidaca, ocitana_lokacija
                    LOAD    stanje_brojaca, 1
                    STORE   stanje_brojaca, brojac_znamenki
                    LOAD    s0, sF
                    CALL    ispisi_s0
                    JUMP    zavrsi_ISR
mod_ucitaj_data :   COMP    stanje_brojaca, 2
                    JUMP    NZ, nibl_2
                    CALL    LCD_Reset
                    CALL    disp_Data_por
nibl_3:             STORE   stanje_prekidaca, ocitani_podatak10to8
                    LOAD    s0, sF
                    CALL    ispisi_s0
                    JUMP    smanji_brojac
nibl_2:              COMP    stanje_brojaca, 1
                    JUMP    NZ, nibl_1
                    SL0     stanje_prekidaca
                    SL0     stanje_prekidaca
                    SL0     stanje_prekidaca
                    SL0     stanje_prekidaca
                    STORE   stanje_prekidaca, ocitani_podatak7to0
                    LOAD    s0, sF
                    CALL    ispisi_s0
                    JUMP    smanji_brojac
nibl_1:             FETCH   s0, ocitani_podatak7to0
                    OR      stanje_prekidaca, s0
                    STORE   stanje_prekidaca, ocitani_podatak7to0
                    LOAD    stanje_brojaca, 2
                    STORE   stanje_brojaca, brojac_znamenki
                    LOAD    s0, sF
                    CALL    ispisi_s0
                    JUMP    zavrsi_ISR
```

```
smanji_brojac:      SUB     stanje_brojaca, 1
                    STORE   stanje_brojaca, brojac_znamenki
                    JUMP    zavrsi_ISR
provjeri_btn_south: COMP    stanje_gumba, 2                 ;citaj
                    JUMP    NZ, provjeri_btn_north
                    CALL    LCD_reset
                    CALL    disp_trazeni_por
                    FETCH   adresa, ocitana_lokacija
                    OUT     adresa, Mem_Adr
                    OUT     adresa, CitajMem
                    CALL    zakasni_1us
                    IN      podatak, Mem_Data_10to8
                    LOAD    s0, podatak
                    AND     s0, 7
                    CALL    ispisi_s0
                    IN      podatak, Mem_Data_7to0
                    LOAD    s0, podatak
                    SR0     s0
                    SR0     s0
                    SR0     s0
                    SR0     s0
                    CALL    ispisi_s0
                    LOAD    s0, podatak
                    AND     s0, 15
                    CALL    ispisi_s0
                    JUMP    zavrsi_ISR
provjeri_btn_north: COMP    stanje_gumba, 4                 ;pisi
                    JUMP    NZ, provjeri_btn_east
                    CALL    LCD_reset
                    CALL    disp_OK_por
                    FETCH   podatak, ocitani_podatak7to0
                    OUT     podatak, Mem_Data_7to0
                    FETCH   podatak, ocitani_podatak10to8
                    OUT     podatak, Mem_Data_10to8
                    FETCH   adresa, ocitana_lokacija
                    OUT     adresa, Mem_Adr
                    OUT     adresa, PisiMem
                    CALL    zakasni_1us
                    JUMP    zavrsi ISR
```

```
provjeri_btn_east : COMP      stanje_gumba, 8                      ;upis adrese
                    JUMP      NZ, provjeri_btn_rot
                    LOAD      stanje_moda_upisa, 1
                    STORE     stanje_moda_upisa, mod_upisa
                    LOAD      stanje_brojaca, 1
                    STORE     stanje_brojaca, brojac_znamenki
                    CALL      LCD_Reset
                    CALL      disp_Addr_por
                    JUMP      zavrsi_ISR
provjeri_btn_rot:   COMP      stanje_gumba, 16                     ;upis podatka
                    JUMP      NZ, provjeri_btn_rot_t
                    LOAD      stanje_moda_upisa, 0
                    STORE     stanje_moda_upisa, mod_upisa
                    LOAD      stanje_brojaca, 2
                    STORE     stanje_brojaca, brojac_znamenki
                    CALL      LCD_Reset
                    CALL      disp_Data_por
                    JUMP      zavrsi_ISR
provjeri_btn_rot_t: COMP      stanje_gumba, 0                   ;promjena stanja led-ice
                    JUMP      NZ, zavrsi_ISR
                    FETCH     stanje_moda_upisa, kcpsm_led
                    COMP      stanje_moda_upisa, $FF
                    JUMP      NZ, KCPSM_ukljuci
                    LOAD      stanje_moda_upisa, 0
                    CALL      LCD_Reset
                    CALL      disp_A2CPU_por
                    OUT       stanje_moda_upisa, KCPSM3_led_port
                    JUMP      dalje
KCPSM_ukljuci:      LOAD      stanje_moda_upisa, $FF
                    OUT       stanje_moda_upisa, KCPSM3_led_port
                    CALL      LCD_Reset
                    CALL      disp_KCPSM3_por
dalje:              STORE     stanje_moda_upisa, kcpsm_led
zavrsi_ISR:         LOAD      s6, 250
                    CALL      zakasni_Nms
                    RETI      ENABLE


; ***************************************************************************************
; Prekidni vektor
; ***************************************************************************************

                    ORG       $3FF
                    JUMP      ISR
                    VHDL      "Rom_Form.vhd", "ProgRom.vhd", "ProgRom"
```

4.3.1.3 VHDL kod povezanog sustava

U sljedećem se listingu navodi kod sustava sa sheme na 32. stranici. Na toj se shemi vide i dva dodatna sklopa: ispravljač signala za gumb BTN North, te djeljitelj signala takta. Naime, dok pritisak na ostale gumbe ne mora biti „ispeglan“, signal kojeg generira pritisak na BTN North dok je aktivan A2CPU mora, i to radi komponenta debouncer. Dok je aktivan KCPSM3, ispravljanje signala se vrši softverski. Djeljitelj takta pak dijeli osnovni takt na ploči (50 Mhz), sa brojem koji je u kodu određen konstantom divider koja je namještena na 5000000, što znači da A2CPU radi na 10Hz. Maksimalan takt (utvrđeno testovima) za koji A2CPU još uvijek radi dobro je 25Mhz.

```
library ieee;
library A2CPU_lib;
library MijoCustomLib;
use ieee.std_logic_1164.all;

entity A2CPU_testsint is
  port(
    strataflash_oe : out std_logic;
    strataflash_ce : out std_logic;
    strataflash_we : out std_logic;
    switch_in : in std_logic_vector(3 downto 0);
    btn_west : in std_logic;
    btn_south : in std_logic;
    btn_north : in std_logic;
    btn_east : in std_logic;
    btn_rot : in std_logic;
    btn_rot_a : in std_logic;
    btn_rot_b : in std_logic;
    KCPSM3_LED_p : out std_logic;
    A2CPU_start_led_p : out std_logic;
    A2CPU_stop_led_p : out std_logic;
    A2CPU_reset_led_p : out std_logic;
    A2CPU_enable_led_p : out std_logic;
    A2CPU_clk_led_p : out std_logic;
    lcd_d : out std_logic_vector(7 downto 4);
    lcd_rs : out std_logic;
    lcd_rw : out std_logic;
    lcd_e : out std_logic;
    clk : in std_logic
  );
end A2CPU_testsint;
```

```
architecture Behavioral of A2CPU_testsint is
  constant divider : integer := 5000000;  --0,1 sec @ clock = 50 MHz
  constant btn_north_debouncetime : integer := 15000000;  --0,3 sec @ clock = 50 MHz
  signal rst_s : std_logic := '1';
  signal reset_A2CPU_s : std_logic := '0';
  signal enable_A2CPU_s : std_logic := '0';
  signal start_A2CPU_s : std_logic := '0';
  signal stop_A2CPU_s : std_logic := '0';
  signal A2CPU_InMem_s : std_logic_vector(10 downto 0);
  signal A2CPU_OutMem_s : std_logic_vector(10 downto 0);
  signal A2CPU_AddrMem_s : std_logic_vector(7 downto 0);
  signal A2CPU_WriteMem_s : std_logic;
  signal A2CPU_ReadMem_s : std_logic;
  signal A2CPU_clk_s : std_logic;
  signal show_A2CPU_clk_s : std_logic := '0';
  signal btn_north_deb_s : std_logic_vector(0 downto 0) := "0";
  signal btn_north_vec_s : std_logic_vector(0 downto 0) := "0";
begin

  clock_divider:entity MijoCustomLib.clock_divider
    generic map(divider)
    port map(reset_A2CPU_s,clk,A2CPU_clk_s);

  btn_north_vec_s(0) <= btn_north;

  btn_north_debouncer:entity MijoCustomLib.generic_debouncer
    generic map(btn_north_debouncetime,1)
    port map(rst_s,clk,btn_north_vec_s,btn_north_deb_s);

  reset_kcpsm3:process(clk)
    variable count_clk:integer := 0;
  begin
    if rst_s = '1' then
      if rising_edge(clk) then
        count_clk := count_clk + 1;
          if count_clk > 100 then
              rst_s <= '0';
            end if;
       end if;
    end if;
  end process;
```

```
  pomocni_KCPSM3: entity A2CPU_lib.pomocni
    port map(strataflash_oe,strataflash_ce,strataflash_we,
             switch_in,btn_west,btn_south,btn_north,btn_east,btn_rot,btn_rot_a,
              btn_rot_b,KCPSM3_LED_p,lcd_d,lcd_rs,lcd_rw,lcd_e,clk,rst_s,enable_A2CPU_s,
               A2CPU_clk_s,A2CPU_OutMem_s,A2CPU_InMem_s,A2CPU_AddrMem_s,A2CPU_WriteMem_s,
              A2CPU_ReadMem_s);

  reset_A2CPU_s <= btn_west when (enable_A2CPU_s = '1') else '0';
  start_A2CPU_s <= btn_south when (enable_A2CPU_s = '1') else '0';
  stop_A2CPU_s <= btn_east when (enable_A2CPU_s = '1') else '0';

  A2CPU_reset_led_p <= reset_A2CPU_s;
  A2CPU_start_led_p <= start_A2CPU_s;
  A2CPU_stop_led_p <= stop_A2CPU_s;
  A2CPU_enable_led_p <= enable_A2CPU_s;

  process(btn_north_deb_s(0))begin
    if rising_edge(btn_north_deb_s(0))then
      if enable_A2CPU_s = '1' then
        show_A2CPU_clk_s <= not show_A2CPU_clk_s;
      end if;
    end if;
  end process;
A2CPU_clk_led_p <= A2CPU_clk_s when(show_A2CPU_clk_s = '1') else '0';

  A2CPU:entity A2CPU_lib.Simple_CPU
    generic map(11,8,3,13,8,8)
    port map(A2CPU_clk_s,reset_A2CPU_s,start_A2CPU_s,stop_A2CPU_s,A2CPU_InMem_s,
             A2CPU_ReadMem_s,A2CPU_WriteMem_s,A2CPU_OutMem_s,A2CPU_AddrMem_s);

end Behavioral;
```

4.3.2 Način komunikacije dva procesora

KCPSM3 i A2CPU komuniciraju preko zajedničke memorije. Važno je uočiti da zato samo jedan procesor može pristupati memoriji u nekom trenutku. Zbog toga su prekidi onemogućeni dok radi A2CPU, kao što su i memorijske sabirnice od A2CPU onemogućene dok radi KCPSM3. Dok je aktivan KCPSM3, korisnik može pregledavati i mijenjati sadržaj memorije (te na taj način zapravo programirati A2CPU, jer su u zajedničkoj memoriji njegove instrukcije). Kad je aktivan A2CPU, pritiskom na start (BTN South, vidjeti dalje) kreće sljedno izvršavanje instrukcija iz zajedničke memorije počevši s nultom lokacijom.

Početno, zajednička je memorija inicijalizirana u VHDL – u, i sadrži programski odsječak koji se vidi iz slijedećeg listinga :

```
MemField :=
  (  "00000011110",  --ld 30
     "01000011111",  --add 31
     "00100100000",  --st 32
     "00000100001",  --ld 33
     "01100100010",  --and 34
     "00100100011",  --st 35
     "00000100100",  --ld 36
     "11000000000",  --not
     "00100100101",  --st 37
     "00000100111",  --ld 38
     "00000100111", --ld 39
     "11100000000", --sh
     "00100101000", --st 40
     "00000101001", --ld 41
     "01000101010", --add 42
     "10100010100", --jmpz 20
     "00100101110", --st 46
     "00000000000",
     "00000000000",
     "00000000000",
     "00000101011", --ld 43
     "01000101100", --add 44
     "00000000000",
     "00000000000",
     "00000101011", --ld 43
     "01000101100", --add 44
     "10100000000", --jmpz 0
     "00100101101", --st 45
     "10000000000", --jmp 0
     "00100101111", --st 47
     "00000000000",
     "00000000000",
     "00000000000",
     "00000000000",
     ------------------------------------------------
     "00000000010",  --30
     "00000000010",
     "00000000000",  --32
     "00001110101",
     "00010011011",
     "00000000000",  --35
     "11011000011",
     "00000000000",  --37
     "00001111011",
     "00001011110",
     "00000000000",
     "00000000000",
     "00000000001",
     "00000000010",
     "00011001100",  --45
     "00000000111",
```

4.4 Povezivanje sklopa A2CPU_testsint sa stvarnim signalima – ucf datoteka

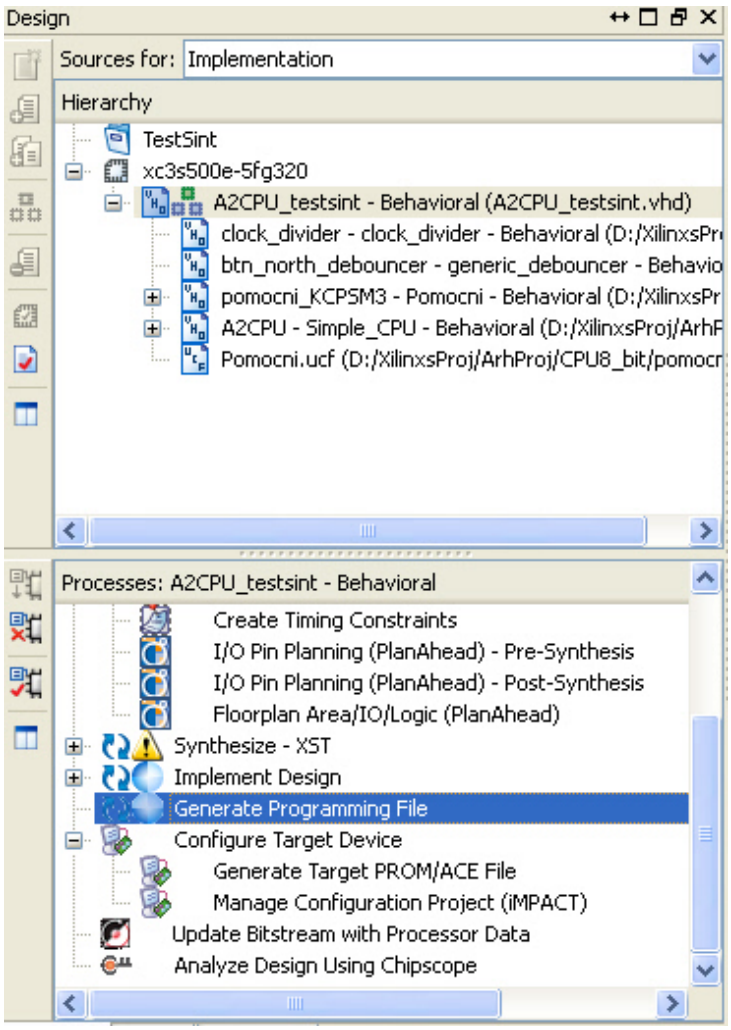
Sklop A2CPU_testsint je u hijerarhiji povezivanja na najvišem nivou – tzv. top level source, i to se može vidjeti iz listinga na strani 41. Njegovi se izvodi (portovi) povezuju s stvarnim izvodima na ploči koji šalju signale generirane od različitih ulaznih jedinica te sa izvodima koji šalju signale različitim izlaznim jedinicama. Povezivanje se vrši pomoću ucf datoteka, a njezin izgled za ovaj projekt dan je u nastavku :

```
NET "clk" LOC = "C9" | IOSTANDARD = LVTTL;
NET "lcd_rs"   LOC = "L18" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_rw"   LOC = "L17" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_e"    LOC = "M18" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<4>" LOC = "R15" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<5>" LOC = "R16" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<6>" LOC = "P17" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<7>" LOC = "M15" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "strataflash_oe" LOC = "C18" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "strataflash_ce" LOC = "D16" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "strataflash_we" LOC = "D17" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 2;
NET "switch_in<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "switch_in<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
NET "switch_in<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;
NET "switch_in<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;
NET "btn_east"  LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "btn_north" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "btn_south" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "btn_west"  LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "btn_rot"   LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "btn_rot_a"   LOC = "K18" | IOSTANDARD = LVTTL | PULLUP ;
NET "btn_rot_b"   LOC = "G18" | IOSTANDARD = LVTTL | PULLUP ;
NET "KCPSM3_LED_p" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "A2CPU_stop_led_p" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "A2CPU_start_led_p" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "A2CPU_reset_led_p" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "A2CPU_enable_led_p" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "A2CPU_clk_led_p" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

Prvo se navodi ime porta komponente (npr. btn_east) a zatim izvod na pločici s kojim se povezuje (H13 za btn_east). Nakon toga se navode neke dodatne opcije koje ti izvodi mogu imati (npr. vrsta otpornika na koju su priključeni – PULLUP ili PULLDOWN).

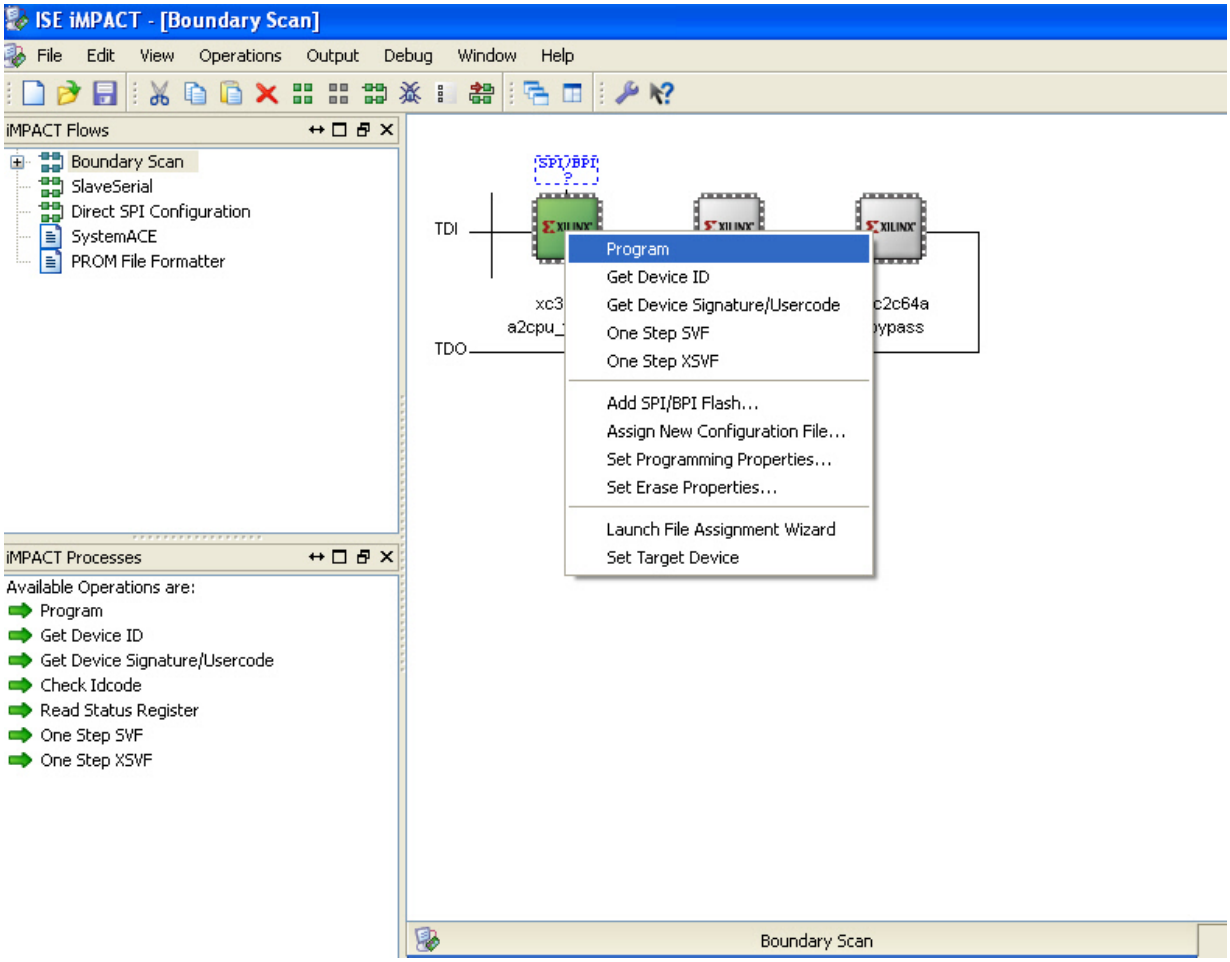
4.5 Uporaba Xilinx alata u sintezi i pretakanje na pločicu

Nakon što je završen VHDL opis projekta, te provedena sinteza (u svrhu provjere može li se čitav VHDL kod može sintetizirati; naime, postoje i VHDL konstrukti koji su samo teoretski, i ne mogu se izvesti sa stvarnim sklopovima) može se pristupiti pretakanju na pločicu. Postupak je jednostavan, a ovdje je prikazan u programskom okruženju Xilinx ISE 11.1:



Potrebno je dakle stisnuti Generate Programming File u listingu procesa raspoloživih za projekt. Time će se generirati datoteka pomoću koje se programira FPGA polje na pločici. Prijenos podataka se obavlja preko USB-a. Nakon generiranja potrebno je odabrati Manage Configuration Project (iMPACT) koji otvara program iMPACT. U

otvorenom programu treba odabrati Boundary Scan, a zatim desnim klikom odabrati Initialize Chain. Nakon toga treba učitati generiranu .bit datoteku i stisnuti desni klik na xasdfasdasdasdad. Odabire se Program i nakon toga je postupak završen – sklop je sada hardverski implementiran u FPGA polju.



Važno je napomenuti da u opcijama projekta (u Xilinx-u Project -> Design Properties) mora pod Device biti odabrana XC3S500E a pod Package FG320 .

Prilikom prvog spajanja pločice preko USB-a windows prepoznaje sve potrebne driver-e (ako je na sistemu instaliran Xilinx).

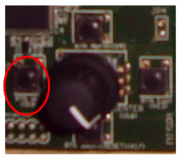
4.6 Opis rada sustava za testiranje procesora

Pošto se pomoću Xilinx-a naš sustav pretoči na pločicu, možemo krenuti s njegovom uporabom. Prikazuje se poruka :

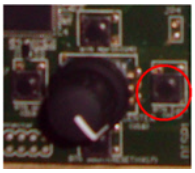
na LCD prikazniku. Početno je A2CPU stopiran, a radi picoBlaze (KCPSM3). Pogledajmo čemu služe pojedini vanjski priljučci:

- rotirajući gumb: ima dvije funkcije: njegovim rotiranjem (bilo u lijevu ili desnu stranu) mijenja se aktivni procesor: A2CPU ili KCPSM3. Druga funkcija mu je omogućena samo kad je aktivan KCPSM3: tada se pritiskom na njega mijenja način upisa u način Podatak (vidjeti dalje)

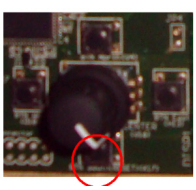
- BTN West: Dok je aktivan KCPSM3, pritiskom na njega se upisuje hex znamenka koju predstavlja položaj kliznih prekidača (vidjeti dalje). Dok je aktivan A2CPU, pritiskom na ovaj gumb resetira se A2CPU.



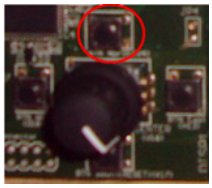
- BTN East: Dok je aktivan KCPSM3, pritiskom na njega se prebacuje u način Upis Adrese (na LCD-u piše Adresa:). Dok je aktivan A2CPU, služi za privremeno zaustavljanje procesora (ne reset !)



- BTN South: Dok je aktivan KCPSM3, pritiskom na njega se na LCD-u prikazuje podatak koji se nalazi na traženoj adresi (na LCD-u piše Trazeni Podatak:). Dok je aktivan A2CPU, služi za pokretanje procesora.

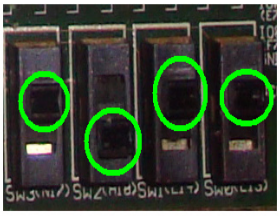


- BTN North: Dok je aktivan KCPSM3, pritiskom na njega se upisani podatak



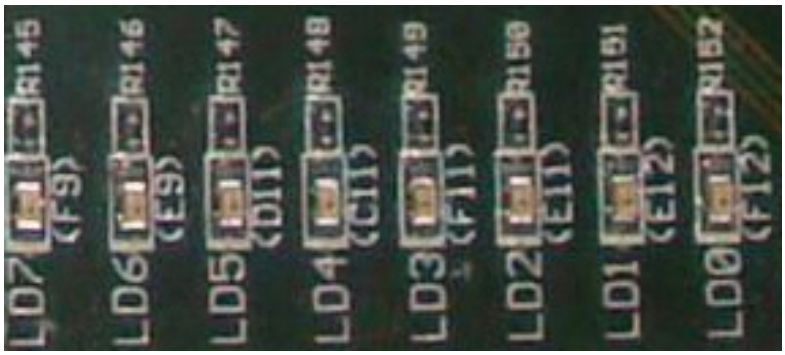
upisuje u zajedničku memoriju (na LCD-u piše Podatak upisan:). Dok je aktivan A2CPU, uključuje/isključuje prikazivanje takta na A2CPU (vidjeti dalje);

- Klizni prekidači: Dok je aktivan KCPSM3, njihov položaj određuje hex



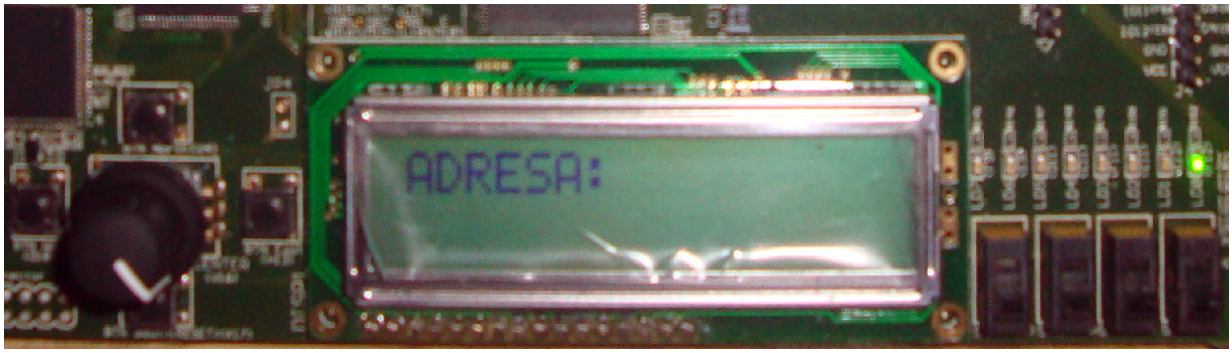
znamenku koja će se upisati u memoriju procesora KCPSM3 (tzv. ScratchPad memory, vidjeti skicu na str.30) i ujedno ispisati na LCD-u. Tako položaj prekidača na slici lijevo predstavlja hex znamenku B.

- Led diode: LD0 gori ako je aktivan KCPSM3, LD7 ako je aktivan A2CPU. LD6 prikazuje taktove A2CPU (ipak, to se može primjetiti ako je broj taktova u sekundi nizak, npr 10 u sekundi).LD1 prikazuje reset A2CPU (odnosno, pritisak na BTN West). Također, LD2 prikazuje start, a LD3 stop procesora A2CPU. Konkretan raspored dioda vidi se na slici :



4.7 Primjer rada

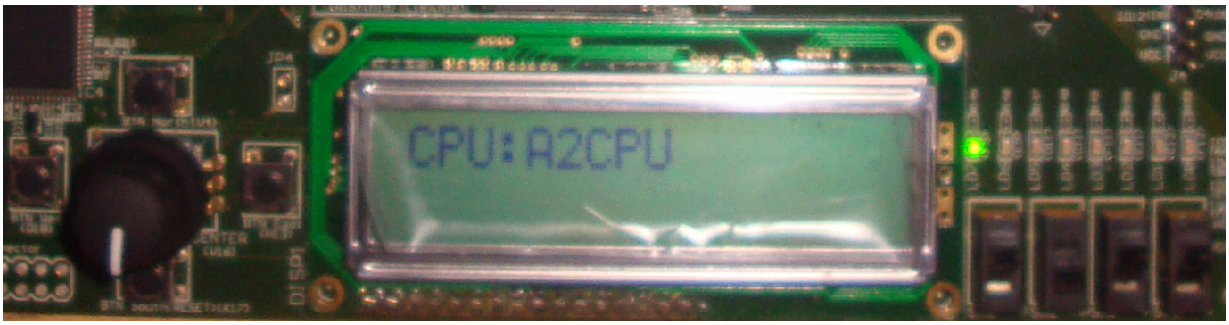
Kao što je rečeno, nakon programiranja na LCD-u se pojavljuje poruka :



Također, gori dioda LD0, što znači da je aktivan KCPSM3

Sa listinga na strani 44 vidi se da odsječak koda za A2CPU, između ostalog, zbraja brojeve na adresama 30 (1E) i 31 (1F) i sprema rezultat na adresu 32 (20) (to se vidi iz prve tri instrukcije: ld 30, add 31, st 32). Početno je na adresama 1E i 1F zapisan broj 002, a na 20 broj 0. Dakle, ako A2CPU radi kako treba, nakon njegova pokretanja na adresi 20 treba pisati broj 004.

Da bi pokrenuli A2CPU, zarotiramo rotirajući gumb. Prikazuje se poruka, i gori LD7 :



Sada pritisnemo BTN West (reset A2CPU) a zatim BTN South (start A2CPU) . Ako je takt nizak (u kodu je postavljen na 10 Hz) ,trebamo pričekati par sekundi da se izvede cijeli odsječak (ili barem prve tri instrukcije). Nakon toga ponovno okrenemo rotirajući gumb (prije toga opcionalno se može zaustaviti A2CPU

pritiskom na BTN East, ali nije nužno) i sada je aktivan KCPSM3 :

Sada pritisnemo BTN East i pojavljuje se poruka Adresa. Upisujemo adresu pomoću kliznih prekidača i gumba BTN West, na način da prvo postavimo prekidače u položaj koji odgovara binarnom kodu znamenke koju upisujemo, a zatim pritisnemo BTN West. Postupak ponavljamo 2 puta, jer trebamo upisati dvije hex znamenke (memorija ima dubinu 255).

U našem slučaju upisujemo adresu 20 :

Sada pritisnemo gumb BTN South i na ekranu se ispisuje :

Dakle, rezultat je očekivan.

Pokušajmo sada isprogramirati A2CPU, odnosno izmjeniti odsječak koji izvodi. Upisat ćemo na adresu 1E podatak 005 a na adrsu 1F podatak 009. Također, neka sada početak programskog odsječka glasi ovako :

```
ld 30;  and 31; st 32;
```

Izmjena je dakle što sada ne zbrajamo podatke na adresama 1E i 1F već ćemo izvršiti operaciju AND nad njima. Rezultat spremamo i dalje na adresu 20, ali on sada treba biti 0001 (1001 AND 0101 = 0001).

Prvo ćemo upisati podatak 005 na adresu 1E. Pritisnemo gumb BTN East i unosimo adresu :

Zatim pritisnemo rotacijski gumb (ne rotiramo!) i prikazuje se poruka :

Na isti način na koji smo unjeli adresu, sada unosimo podatak 005 :

Nakon što smo unjeli podatak, pritisnemo BTN North i prikazuje se poruka :

To znači da smo na adresu 1E unjeli podatak 005. Isti postupak ponovimo i za adresu 1F na koju upisujemo 009 i adresu 01 na koju upisujemo podatak 31F (to je instrukcija and 31, budući je operacijski kod instrukcije AND jednak 0011 = 3) .

Nakon što smo izmjenili sadržaj memorije, A2CPU učinimo aktivnim rotiranjem gumba, i pokrenemo ga kao što je već bilo opisano. Vratimo se ponovo u stanje u kojem je aktivan KCPSM3, i sada nas zanima stanje na adresi 20. Upišemo tu adresu, pritisnemo BTN South (sve isto kao i u prvom primjeru) i dobijemo:

Rezultat je opet u skladu s očekivanim. A2CPU radi dobro ☺.

Napomena : maksimalan takt (dobiven testiranjem) za A2CPU je 25 Mhz. Ipak, to se može poboljšati, ponajprije boljom izvedbom memorije.

5. Literatura

Arhitektura računala 2, predavanja FER Zagreb 2009

Šegvić S., Uvod u programski jezik VHDL, Zagreb 2002

Pong P. Chu, FPGA prototyping by VHDL examples

Xilinx Inc.,Spartan 3E manual, 2006

Xilinx Inc.,PicoBlaze manual, 2006

www.google.com

www.wikipedia.org

www.xilinx.com