Elektronski Fakultet u Nišu
Katedra za elektroniku
Predmet: Embeded sistemi

DMA kontroler

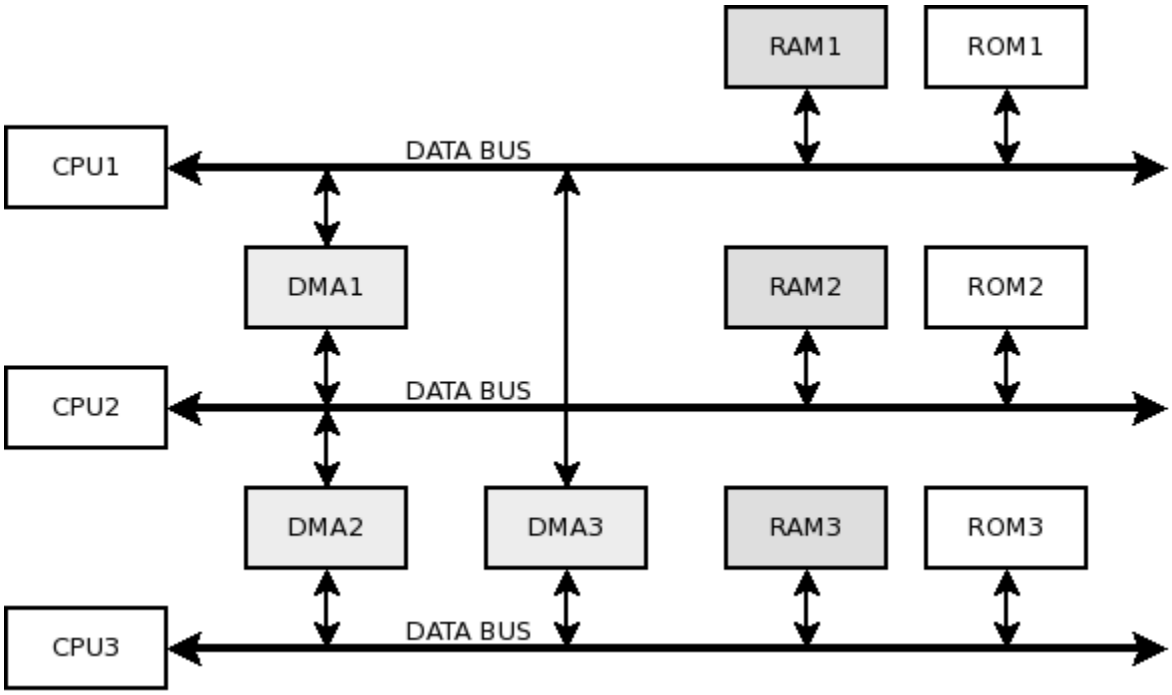
Mentor:prof.dr. Mile Stojčev

Studenti: Igor Stojanović 12777
Nenad Radulović 13152
Miloš Lazić 13166

Sadržaj

1.Uvod

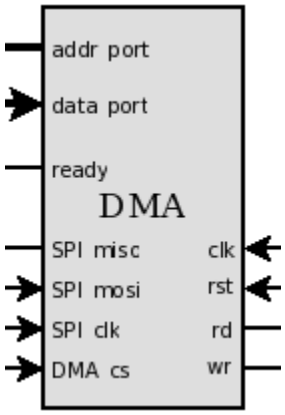
DMA kontroleri su uređaji koji imaju široku primenu u digitalnim sistemima. Njihova uloga je u obavljanju prenosa podataka između različitih memorijskih lokacija, jednog ili više različitih modula. Na ovaj način se procesor oslobađa od upravljanja prenosom podataka, čime se dobija znatna ušteda *CPU*-ovog vremena koje se može iskoristiti za izvršavanje drugih zadataka. Jedna od mogućnosti korišćenja DMA kontrolera je u multiprocesorskim sistemima za razmenu podataka između procesora. U takvim sistemima procesori komuniciraju preko *Dual-port RAM* memorija pri čemu se prenos podataka između tih memorija vrši pomoću *DMA* kontrolera. Procesor iniciran prenos slanjem zahteva *DMA* kontroleru i omogćava mu pravo pristupa adresnoj magistrali i magistrali podataka. *DMA* kontroler, posle izvršenog prenosa obaveštava procesor o obavljenom transferu i predje mu pravo upravljanja nad magistralama. Na slici 1.1 prikazana je šema jednog multiprocesorskog sistema sa *DMA* kontrolerom.



Slika1.1 Multiprocesorski sistem

2. Interfejs

Cilj ovog zadatak je projektovanje *DMA* kontrolera čija je uloga transfer podataka između dve *Dual-port RAM* memorije u multiprocesorskom sisitemu. Na slici 2.1 prikazan je interfejs DAM kontrolera koji je realizovan.

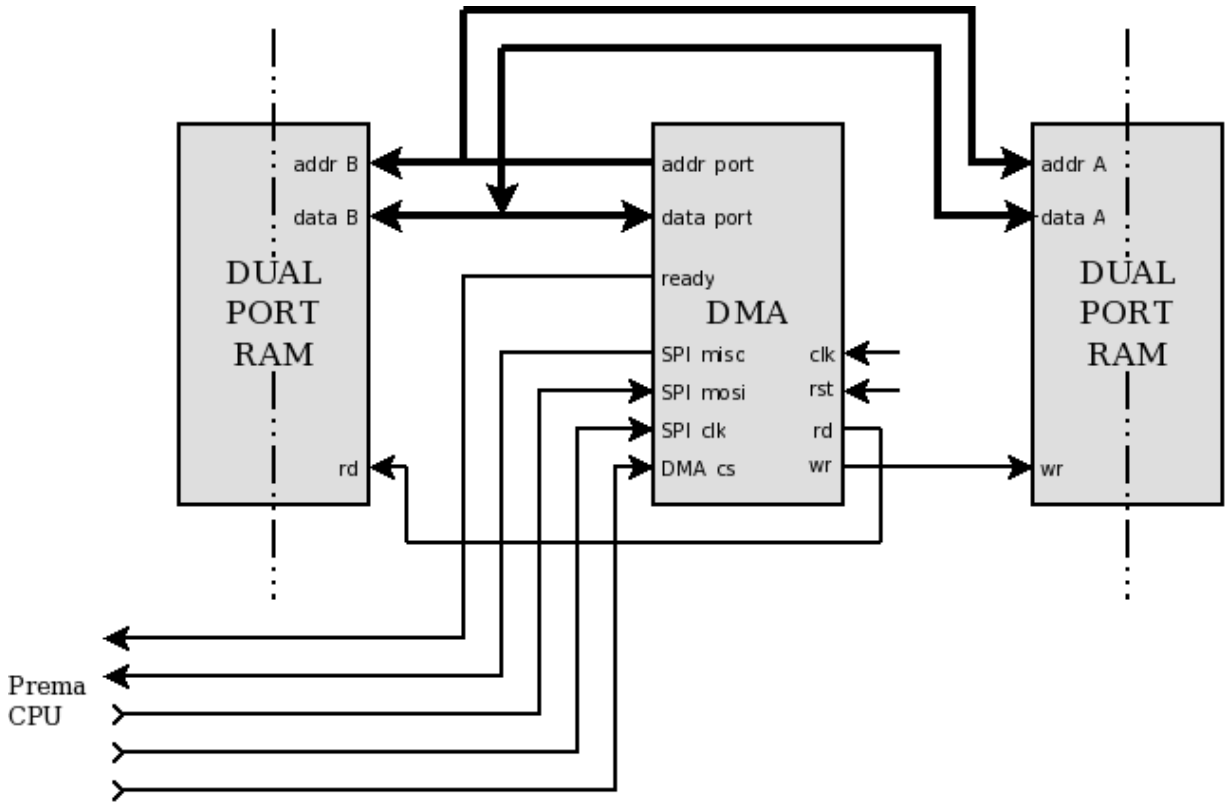


Slika 2.1 Interfejs DMA kontrolera

Na slici se uočavaju sledeći portovi za komunikaciju sa periferijom:

Spisak pinova:

- *CLK*- ulazni signal za taktovanje sistema
- *RST*- ulazni signal za resetovanje sistema
- *SPI mosi*- ulazni signal za serijski prenos podataka od procesora ka DMA kontroleru. Preko ovog porta se vrši upis podatak u interne registre kontrolera
- *SPI miso*- izlazni signal za prenos podatak od DMA ka procesoru
- *SPI clk*- signal takta za serijsku komunikaciju
- *DMA cs*- Ulazni siganl za inicijalizaciju DMA prenosa. Za vreme SPI komunikacje sa procesorom ovaj signal je aktivan. Inicijalizacija se vrši opadajućom ivicom signala, tj. nakon upisa podataka u interne registre
- *ADDR port*- osmobitni izlazni port za povazivanje sa adresnom magistralom
- *DATA port*- osmobitni bidirekcioni port za prenos podataka
- *RD*- signal za iniciranje čitanja iz memorije
- *WR*- signal za iniciranje upisa u memoriju
- *READY*- siganal za pružanje informacija o statusu *DMA* kontrolera. Kada je na nivou logičke nule vrši se *DMA* prenos.



Slika 2.2 Blok šema DMA kontrolera u radnom okruženju

Princip rad DMA kontrolera:

Kada CPU želi da obavi transfer podataka između dual-port RAM memorija, aktivira signal DMA CS nakon čega se startuje serijska komunikacija-SPI. Na ovaj način CPU vrši upis podataka u interne registre kontrolera. Sadržaj tih podataka definiše broj bajtova po kanalu koji se treba preneti i način generisanja adresa za odredišnu dual-port RAM memoriju . Dok je aktivna SPI komunikacija signal DMA CS je aktivan. Nakon prenosa poslednjeg bita podatka signal se deaktivira što ujedno predstavlja inicijalizaciju DMA transfera. DMA kontroler prvo čita podatak iz izvorišnog dual-port RAM-a postavljanjem adrese na odgovarajuću magistralu i aktiviranjem signala RD kojim obavestava memoriju da zeli da obavi ciklus citanja. Kada memorija postavi podatak adresirane lokacije, kontroler ga prihvata, smešta u interni bafer, a zatim generiše adresu odredišne memorijske lokacije u kojoj će se smestiti podatak iz bafera i aktivira signal WR kojim obaveštava odredišnu memoriju da želi da obavi ciklus čitanja. Generisanje adrese se vrši po algoritmu definisanim od strane CPU-a upisom odgovarajućih podataka u interne registre kontrolera. Proces se ponavlja dok se ne prebaci celokupan broj specificiranih bajtova. Nakon obavljenog prenosa DMA kontroler aktivira signal READY čime se obaveštava CPU.

3. Struktura DMA kontrolera

DMA kontroler ima zadatak da čita podatke iz izvorišne memorije i prebacuje u odredišnu meoriju. Podaci su u izvorišnoj memoriji smešteni u memorijskim lokacijama sekvencijalno poređanim u jednom nizu. Ti podaci se u odredišnoj memoriji smeštaju po sledećem algoritmu:

$(odredišna\_adresa)_i = i_i * SB + C_i + B_i$.

- i_i- indeks, inkrementira se nakon svakog upisa u odredišnu lokaciju.
- SB- definiše faktor multiplikacije po sledećem zakonu:

SB	faktor multiplikacije
00	x1
01	x2
10	x4
11	x1

Tabela 1.

- C_i- konstanta definisana u hardveru.

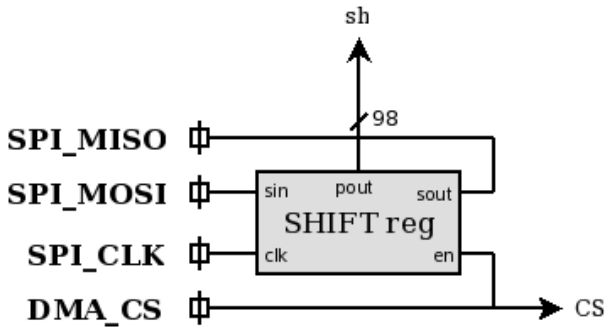
C_i	vrednost
0	01h
1	04h
2	10h
3	80h

Tabela 2.

- B_i- bazna adresa

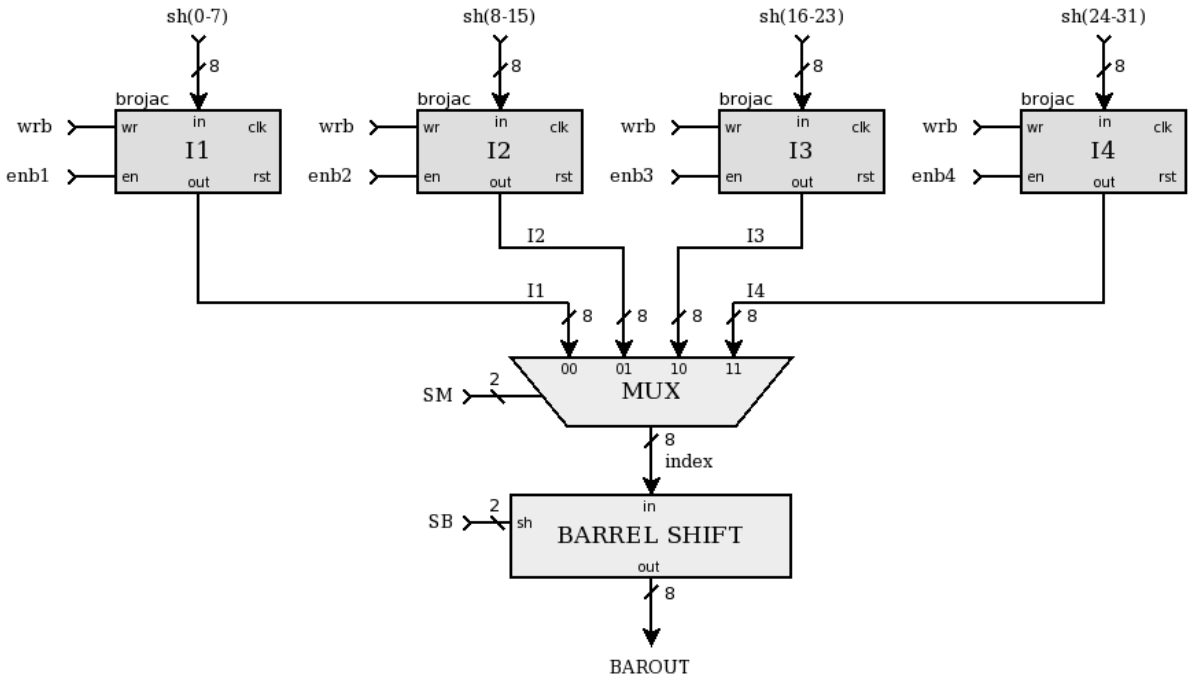
Staza podataka

U daljem tekstu biće posvećena pažnja opisu staze podataka DMA kontrolera.



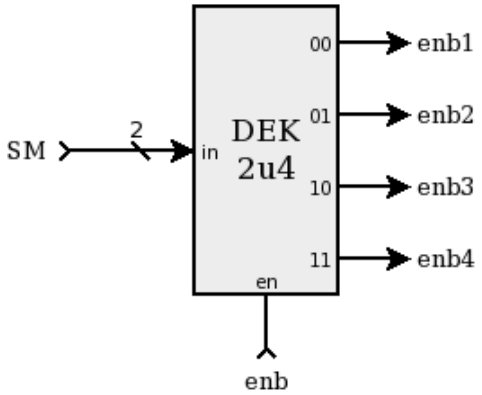
Slika 3.1 98-bitni pomerački registar

Na slici 3.1 je prikazan pomerački registar u koji se, preko SPI komunikacije, upisuju podaci kojima se definišu DMA kanali. Svaki DMA kanal se definiše, faktorom multiplikacije, brojem bajtova koje treba preneti,baznom adresom i indeksnom adresom u odredišnoj memoriji. Upis inicijalnih vrednosti se vrši sekvencijalno za sva četiri kanala. Domaćin procesor postavlja stanje aktivno na pin DMA_CS koji ujedno predstavlja signal dozvole ***en*** za pomerački registar i signal CS kojim se vrši inicijalizacija upravljačke jedinice DMA kontrolera.SPI_CLK je taktni signal koji generiše domaćin procesor pri čemu se na svaku uzlaznu ivicu vrši punjenje pomeračkog registra sa vrednošću koja je prisutna na SPI_MOSI ulaznom pinu. SPI_MISO izlazni pin se koristi kada je potrebno da se ulančuju više SPI modula. Pomerački registar poseduje paralelni izlaz ***sh.***



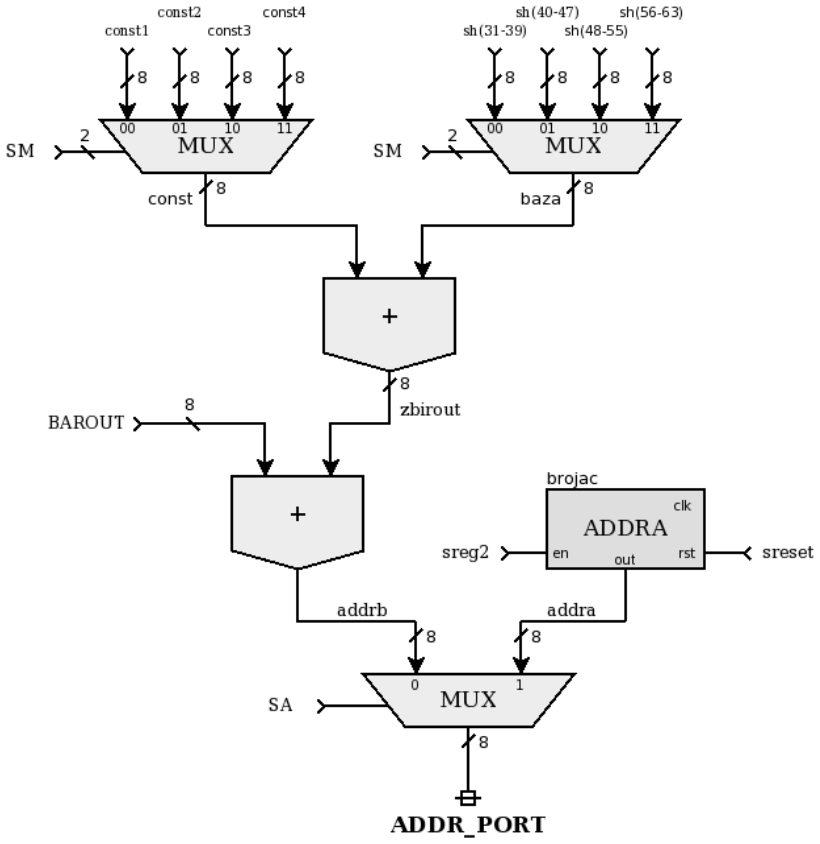
Slika 3.2 Staza podataka-čuvanje indeksa i multiplikacija

Na slici 3.2 je prikazano kako se vrši čuvanje indeksa i_i, njihovo inkrementiranje i multiplikacija sa faktorom SB. Brojači I1 do I4 su brojači sa mogućošću upisa kada je wr=1. Njihovi ulazi za upis povezani su na paralelni izlaz pomeračkog registra ***sh*** sa slike 3.1. Kada je ***wrb*** signal u stanju visoko, onda se vrši upis vrednosti iz pomeračkog registra u brojače I1 do I4. Signalom ***SM*** se bira kanal pomoću multipleksera i vodi na barrel pomerački registar koji vrši propuštanje ili pomeranje signala indeks za jednu odnosno dve bitske pozicije definisane tabelom 1



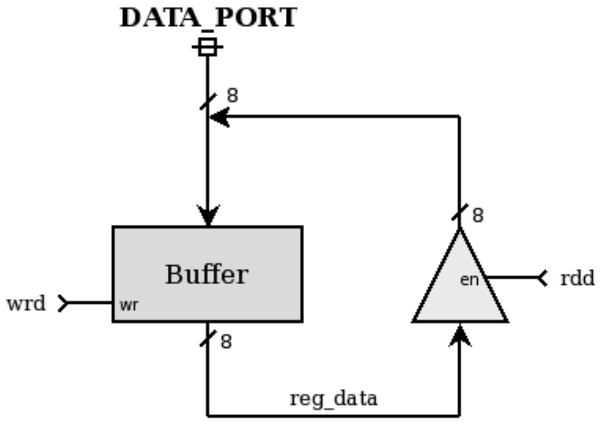
Slika 3.3 Dekoder za indeksne brojače

Dekoderom 2u4 vrši se generisanje signala za dozvolu rada indeksnih brojača. Upravljačka jedinica kontroliše brojanje indeksnih brojača pomoću signala ***enb*** koji ujedno signal dozvole za dekoder.



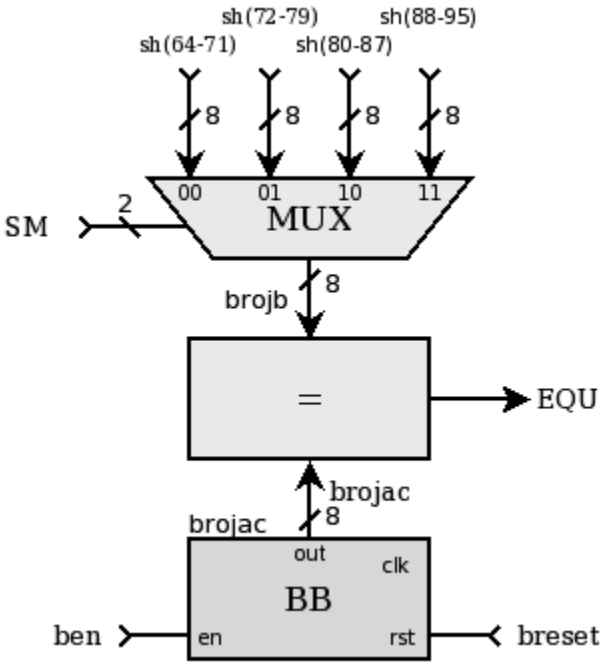
Slika 3.4 Staza podataka-generator adresa

Isto kao u predhodnom slučaju signalom ***SM*** se bira DMA kanal multiplekserima *const* i *baza*. Izlazi multipleksera se vode na sabirač čija se vrednost sa izlaza sabira sa ***BAROUT***. Signalom ***SA*** bira se odredišna ili izvorišna adresa. Izvorišna adresa se generiše pomoću brojača ***ADDRA***.



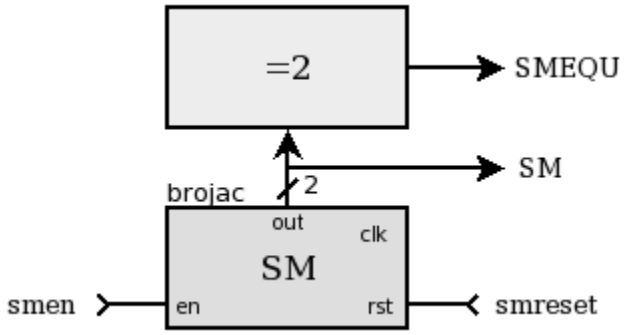
Slika 3.5 Osmobitni bafer za čuvanje podataka

DATA_PORT je bidirekcioni port. Osmobitni bafer čini jedno latch kolo i trostatički osmobitni bafer. Kada je signal ***wrd*** aktivan vrši se upis u latch. Signalom ***rdd*** se dozvoljava propuštanje podataka iz latch-a na ***DATA_PORT***.



Slika 3.6 Staza podataka – brojač bajtova

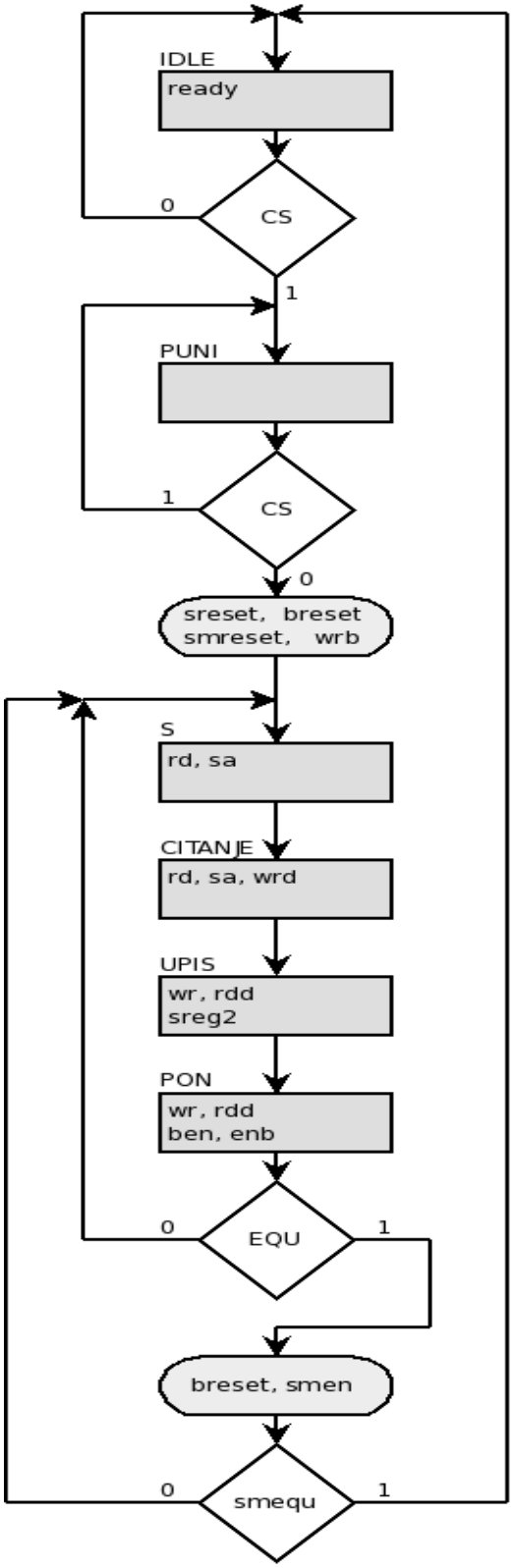
Brojanje prebačenih bajtova se vrši pomoću brojača ***BB*** i komparatora. Komparator vrši poređenje trenutne vrednosti brojača sa vrednošću koja je definisana u vektoru sh. Signal ***SM*** vrši odabir odgovarajuće vrednosti iz vektora ***sh*** tako da svaki kanal DMA prenosa može imati različitu vrednost prenetih bajtova. Kada DMA prenese zadati broj bajtova, signal EQU se postavlja na aktivno stanje čime se signalizira upravljačkoj jedinici da je izvršen prenos zadatih bajtova. Signal ***breset*** se postavalja u aktivno stanje svaki put kada se prelazi na drugi DMA kanal kako bi se njegova vrednost postavila na nulu.



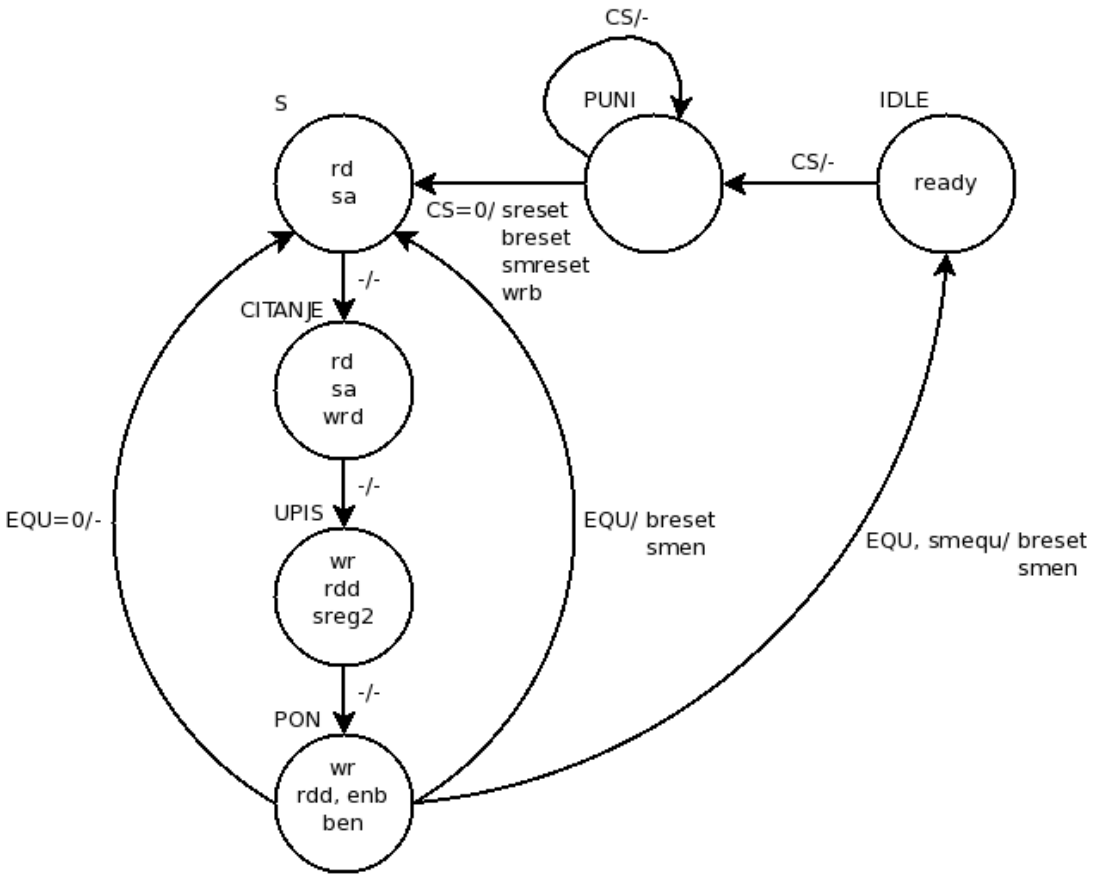
Slika 3.7 Staza podataka – brojač DMA kanala

Za brojač DMA kanala se koristi 2-bitni brojač ***SM***, dok se upravljačkoj jedinici, preko signala ***SMEQU***, signalizira kada je DMA izvršio opsluživanje sva četiri DMA kanala.

Upravljačka jedinica



Slika 3.8a – ASM dijagram upravljačke jedinice



Slika 3.8b Dijagram stanja

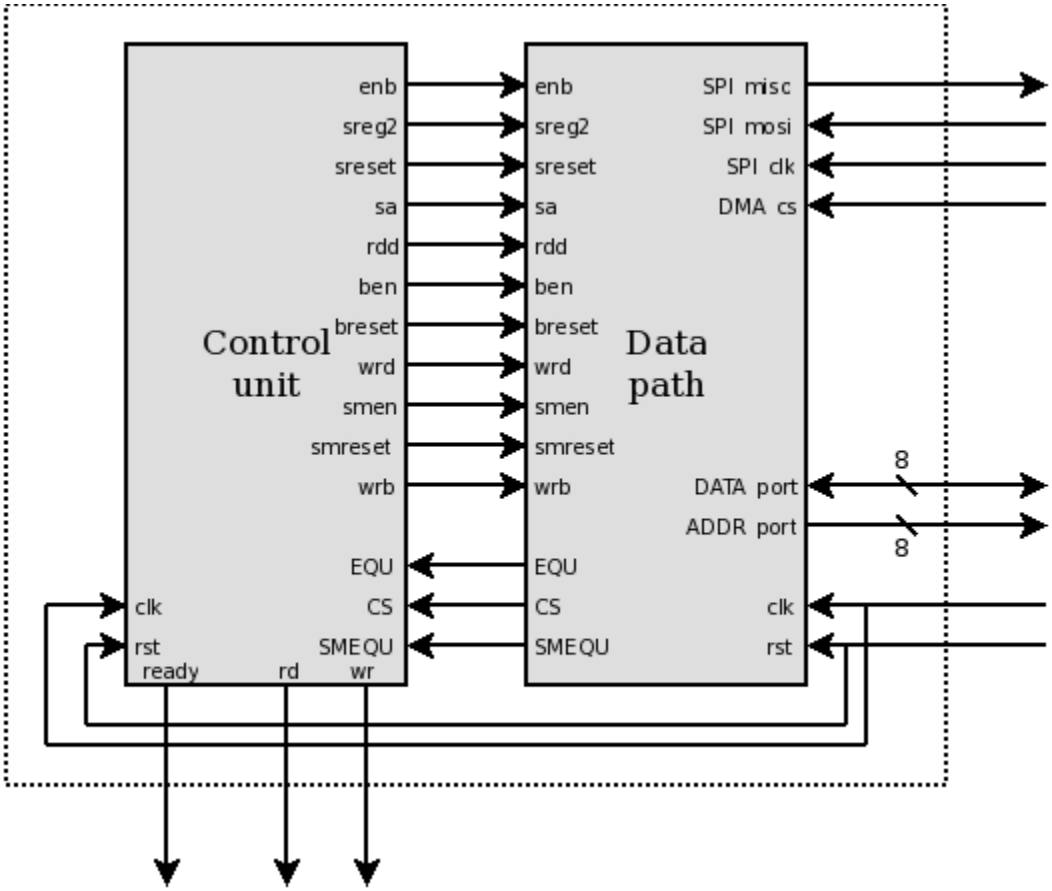
U stanju mirovanja automat se nalazi u stanju IDLE. U tom stanju stalno je aktivan izlazni pin READY koji signalizira domaćinu procesoru da je DMA spreman da prihvati podatke za inicijalizaciju i DMA prenos. U ovom stanju čeka se sve dok se ulazni pin DMA_CS ne postavi u aktivno stanje.

Kada procesor postavi aktivno stanje na DMA_CS pinu (CS=1), radi punjenja pomeračkog registra, automat ulazi u stanje PUNI i nalazi se u tom stanju sve dok se ne završi sa upisom podataka u pomerački registar. Kada je procesor završio sa punjenjem pomeračkog registra onda se DMA_CS pin postavlja na neaktivni nivo (CS=0), što je u stvari signal upravljačkoj jedinici da je procesor završio sa punjenjem i da treba da se pređe na DMA prenos. Pre prelaska u naredno stanje, upravljačka jedinica postavlja kontrolne signale koji vrše resetovanje brojača za generisanje izvorišne adrese, ADDRA, brojača za brojanje DMA kanala, SM i brojača za brojanje bajtova BB i signal za dozvolu upisa u indeksne brojače I1 do I4. U S stanju se pin RD postavlja na aktivno stanje, a signalom SA se bira izvorišna adresa u brojaču ADDRA. Automat prelazi su stanje ČITANJE.

U stanju ČITANJE signali SA i RD ostaju u aktivnom stanju pri čemu se aktivira i signal WRD čime se dozvoljava upis podataka sa magistrale u latch. Automat prelazi u stanje UPIS.

U stanju UPIS se postavlja SA=0 kako bi se postavila odredišna adresa na adresnoj magistrali, signal WR za upis u memoriju i signal RDD radi postavljanja prethodno zapamćenog podatka u na magistralu podataka. Signalom SREG2 se inkrementira brojač izvorišne adrese.

U stanju PON svi signali ostaju nepromenjeni kako bi se obzbedilo dovoljno vremena za obavljanje ciklusa upisa u odredišnu memoriju. Signal BEN se aktivira kako bi se inkrementirao brojač bajtova, dok signal ENB omogućava inkrementiranje indeksnog brojača datog DMA kanala. U ovom stanju se provera da li obavljen celokupni prenos specificiranog broja bajtova datog DMA kanala, a pomoću SMEQU se proverava da li su svi kanali opsluženi. Dozvoljavamo inkrementiranje brojača DMA kanala pomoću signala SMEN i resetujemo brojač bajtova prethodno opsluženog kanala. Ako su svi DMA kanali opsluženi automat se vraća u stanje IDLE u kome postavlja signal READY na aktivno.



Slika 3.9 Povezivanje automata sa stazom podataka.

Izlazni pinov su: ready, rd, wr, addr_port, SPI_miso i koriste se za slanje podataka ka okruženju. Ulazni pinovi su : clk, rst, SPI_mosi, dma_cs, SPI_clk, koriste se za prijem podataka od periferija. Ostali skup signala čine interni signali koji se koriste za spregu kontrolne jedinice i staze podatak.

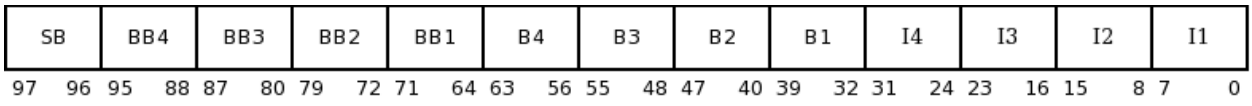
4. Inicijalizacija i korišćenje DMA kontrolera

DMA kontroler ima nakoliko registara dostupnih programeru, i to su:

- ***BB4 do BB1***- osmobitni brojači bajtova po kanalu,
- ***B4 do B1***- osmobitni registri, bazne adrese po kanalu,
- ***I4 do I1***- osmobitni brojači, indeksni brojač po kanalu.

Pored ovih registara programeru je dostupna upotreba dvobitnog parametra, ***SB*** koji određuje faktor multiplikacije za sve kanale.

BB4 do BB1 su parametri koji pokazuju koliko bajtova treba preneti. Vrednost 00h označava da treba da se prenese 256 bajtova. Parametri ***B4 do B1*** i ***I4 do I1*** služe za generisanje odredišne adrese.



Slika 4.1 Format vektora inicijalizacije DMA kontrolera

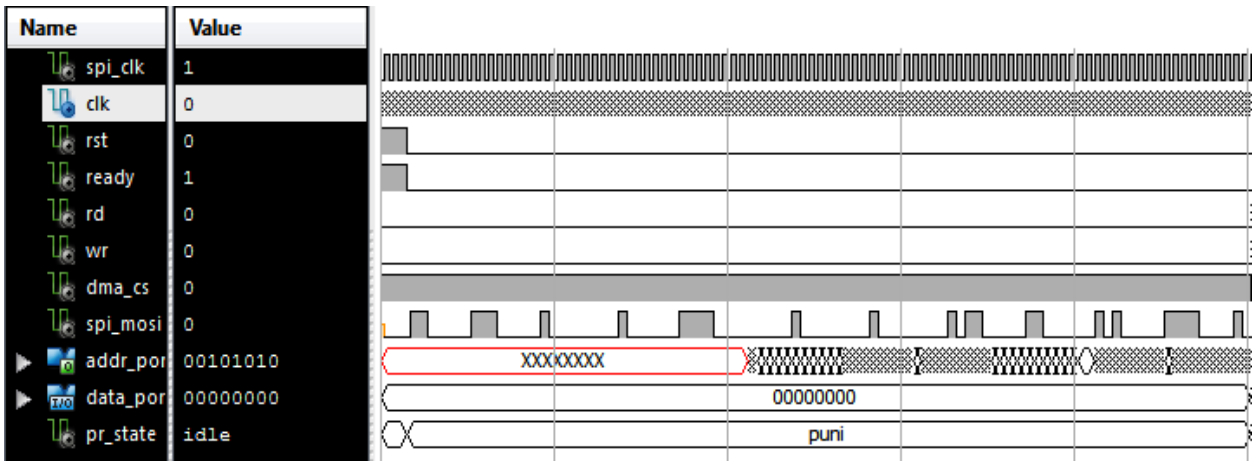
Formula po kojoj se računa odredišna adresa:

$(odredišna\_adresa)_i = i_i * SB + C_i + B_i$.

Na slici 4.1 je prikazan format podatka koji procesor treba da pošalje DMA kontroleru preko SPI komunikacije radi inicijalizacije svih kanala. Nakon inicijalizacije DMA automatski prelazi u režim opsluživanja kanala.

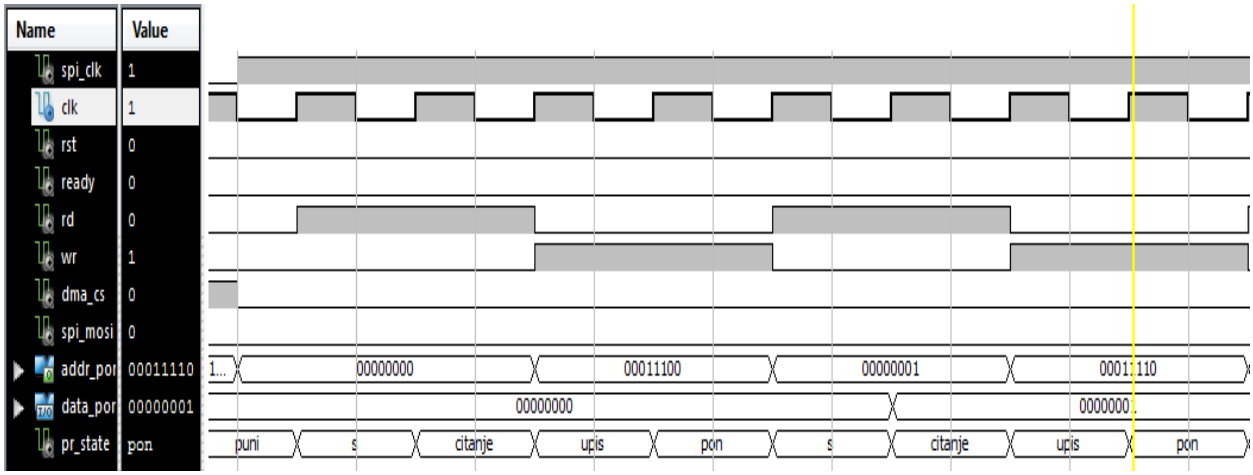
5. Funkcionalna analiza

Nakon razrađivanja algoritama i opisa DMA kontrolera u VHDL kod-u, pristupa se verifikaciji dizajna u ISim simulatoru. Radi verifikacije dizajna, u testbenchu je instaciran ROM popunjen sa prethodno definisanim podacima, koja vrši simulaciju izvorišne Dual Port memorije.



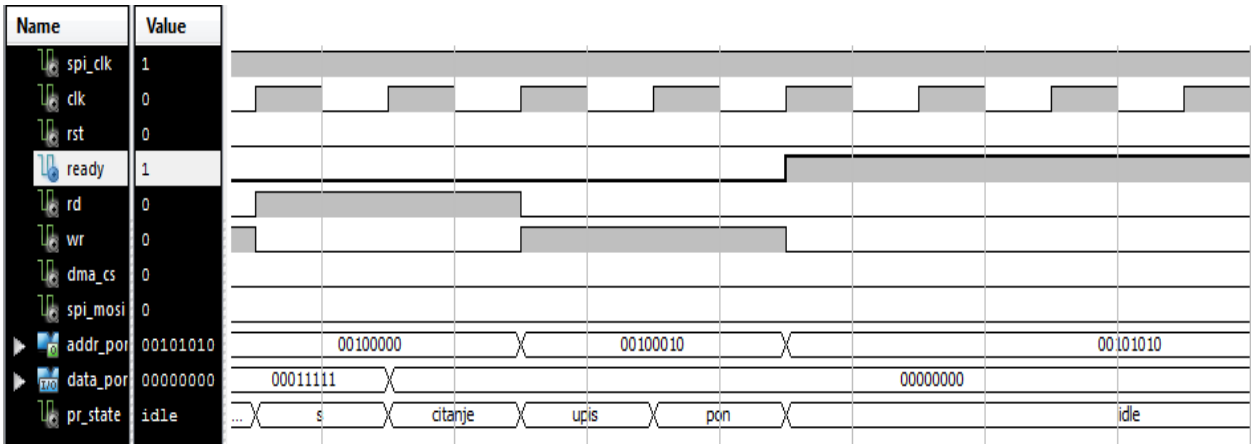
Slika 5.1 Prikaz inicijalizacije DMA kontrolera

Na Slici 5.1 je prikazana vremenska simulacija inicijalizacije DMA kontrolera. Primećuje se aktivnost na ulaznom pinu SPI_MOSI kojim se unose parametri u pomerački registar. DMA_cs mora da bude aktivno za sve vreme SPI upisa. Može se primetiti da je DMA postavio READY izlaz na nisko, signalizirajući da je započeo proces punjenja DMA. PR_state signal je trenutno stanje automata gde se vidi da se automat nalazi u stanju PUNI.



Slika 5.2 Prikaz transfera podataka

Nakon što je DMA inicijalizovan on prelazi u režim transfera podataka. Na gornjoj slici se vidi kako DMA naizmenično generiše RD i WR signale. Svaki proces čitanja traje po dva taktna ciklusa. Kada DMA vrši čitanje izvorišne memorije on prvo postavi adresu na adresnom portu i RD signal. U sledećem taktnom ciklusu vrši se uzorkovanje vrednosti na magistrali podataka i upis u buffer registar. Proces upisa, takođe, traje dva taktna ciklusa. U ovom slučaju DMA postavlja podataka na magistrali podataka i odgovarajuću adresu na adresnoj magistrali.



Slika 5.3 Završetak DMA prenosa

Nakon što je DMA završio sa opsluživanjem svih kanala on postavlja READY signal na aktivno, signalizirajući na taj način da je DMA prenos završen.

6. VHDL opis

Barrel pomerač

Barrel pomerač ima mogućnost da propušta vrednost sa ulaza na izlaz ili da pomera signal za jednu odnosno dve bitske pozicije što je ekvivalentno množenju sa x1, x2 ili x4. Funkciju diktira signal ***sb***.

```
process(index,sb)
begin
      if(sb="00" or sb="11") then
            barout<=index;
      elsif(sb="01") then
            barout<=index(6 downto 0)&'0';
      elsif(sb="10") then
            barout<=index(5 downto 0)&"00";
      end if;
end process;
```

Brojač indeksa

To je binarni brojač sa mogućnošću upisa vrednosti i brojanja po modulu 16.
rst = sinhroni reset,
wr = upis u brojač,
en = dozvola za brojanje.

```
signal output_pom: unsigned(7 downto 0);
process(clk)
begin
      if(clk 'event and clk='1') then
            if(rst='1') then
                  output_pom<=(others=>'0');
            elsif(wr='1') then
                  output_pom<=unsigned(input);
            elsif(en='1') then
                  output_pom<=output_pom+1;
            end if;
      end if;
end process;
output<=std_logic_vector(output_pom);
```

Multiplekser za izbor konstanti

Upravljačkim signalom ***sm*** vrši se izbor izlaznog signala.

```
const<="00000001" when(sm="00") else
       "00000100" when(sm="01") else
       "00010000" when(sm="10") else
       "10000000";
```

Multiplekser za izbor bazne adrese
Ovim multiplekserom bira se bazna adresa u zavisnosti od upravljačkog signala ***sm.***

```
baza<=sh(39 downto 32) when(sm="00") else
      sh(47 downto 40) when(sm="01") else
      sh(55 downto 48) when(sm="10") else
      sh(63 downto 56);
```

Multiplekser za izbor adrese
U zavisnosti od selektorskog signala ***sa*** bira se izvorišna ili odredišna adresa.

```
addr_port<=addra when(sa='1') else
           addrb;
```

Multiplekser za izbor broja bajtova za prenos
Ovim multiplekserom se bira koliko bajtova treba da se prenese za dati kanal.

```
brojb<=sh(71 downto 64) when(sm="00") else
       sh(79 downto 72) when(sm="01") else
       sh(87 downto 80) when(sm="10") else
       sh(95 downto 88);
```

Sabirači

```
--sabirac 1
zbirout<=std_logic_vector(unsigned(const)+unsigned(baza));
--sabirac 2
ukupno<=std_logic_vector(unsigned(barout)+unsigned(zbirout));
addrb<=ukupno;
```

Registar addra za generisanje izvorišne adrese.
sreset = sinhroni reset,
sreg2 = signal dozvole brojanja.

```
process(clk)
begin
      if(clk 'event and clk='1') then
            if(sreset='1') then
                  addra<=(others=>'0');
            elsif(sreg2='1') then
                  addra<=std_logic_vector(unsigned(addra)+1);
            end if;
      end if;
end process;
```

Brojač prenetih bajtova
breset = sinhroni reset,
ben = signal dozvole brojanja.

```
process(clk)
begin
	if(clk 'event and clk='1') then
		if(breset='1') then
			brojac<=(others=>'0');
		elsif(ben='1') then
			brojac<=std_logic_vector(unsigned(brojac)+1);
		end if;
	end if;
end process;
```

Registar buffer
wrd = signal dozvole za upis u registar

```
process(clk)
begin
	if(clk 'event and clk='1') then
		if(wrd='1') then
			reg_data<=data_port;
		end if;
	end if;
end process;
```

Pomerački registar sh
dma_cs = signal dozvole za pomeranje

```
process(spi_clk)
begin
	if(spi_clk 'event and spi_clk='1') then
		if(dma_cs='1') then
			sh<=spi_mosi&sh(97 downto 1);
		end if;
	end if;
end process;
```

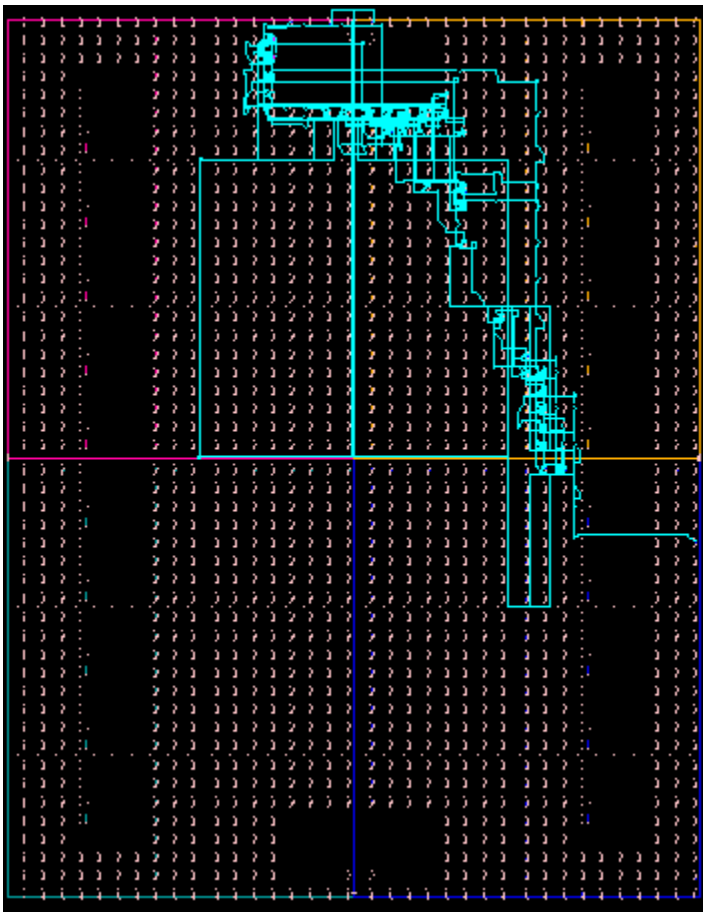
Brojač sm 2-bit
Dvobitni brojač za DMA kanale
smreset = sinhroni reset,
smen = signal dozvole brojanja.

```
process(clk)
begin
      if(clk 'event and clk='1') then
            if(smreset='1') then
                  sm<=(others=>'0');
            elsif(smen='1') then
                  sm<=std_logic_vector(unsigned(sm)+1);
            end if;
      end if;
end process;
```

Dekoder 2u4 sm u enbX
Ovaj dekoder se koristi za generisanje signala dozvole za indeksne registre.
sm = selekcija izlaza,
enb = signala dozvole za izlaz dekodera.

```
dek<="0000" when (enb='0') else
        "0001" when(sm="00") else
        "0010" when(sm="01") else
        "0100" when(sm="10") else
        "1000";
```

7. Implementacija na FPGA razvojnu ploču



Slika 7.1 Prikaz fizičkog rasporeda elemenata DMA kontrolera sa ROM memorijom korišćena u simulaciji

Na slici 7.1 je prikazan raspored elemenata na FPGA čipu, sa vezama koje ih povezuju. FPGA čip je iz familije Spartan3, XC3S500E.

Sintetizovan hardver:

```
4x8-bit ROM                                      : 1
8-bit adder                                      : 2
2-bit up counter                                 : 1
7-bit up counter                                 : 1
8-bit up counter                                 : 6
Flip-Flops                                       : 196
8-bit comparator equal                           : 1
8-bit comparator less                            : 1
1-bit 4-to-1 multiplexer                         : 16
8-bit 4-to-1 multiplexer                         : 2
```

8. Prilog: VHDL kod

Indeksni brojač: brojac_i.vhd

```
----------------------------------------------------------------------------
-- Indeksni brojac
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity brojac_i is
      Port(
            input: in std_logic_vector(7 downto 0);
            wr,clk,rst,en: in std_logic;
            output: out std_logic_vector(7 downto 0));
end brojac_i;

architecture Behavioral of brojac_i is
      signal output_pom: unsigned(7 downto 0);
begin
process(clk)
begin
      if(clk 'event and clk='1') then
            if(rst='1') then
                  output_pom<=(others=>'0');
            elsif(wr='1') then
                  output_pom<=unsigned(input);
            elsif(en='1') then
                  output_pom<=output_pom+1;
            end if;
      end if;
end process;
output<=std_logic_vector(output_pom);

end Behavioral;
```

Barel pomerač: barrel.vhd

```
----------------------------------------------------------------------------
-- Barrel shifter
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity barrel is
      Port(
            index: in std_logic_vector(7 downto 0);
            sb: in std_logic_vector(1 downto 0);
            barout:out std_logic_vector(7 downto 0));
end barrel;

architecture Behavioral of barrel is
begin
process(index,sb)
begin
      if(sb="00" or sb="11") then
            barout<=index;
      elsif(sb="01") then
            barout<=index(6 downto 0)&'0';
      elsif(sb="10") then
            barout<=index(5 downto 0)&"00";
      end if;
end process;

end Behavioral;
```

Staza podataka: staza.vhd

```
----------------------------------------------------------------------------
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity staza is
      Port(
            wrb,spi_clk,enb:in std_logic;
            smen,smreset:in std_logic;
            sreg2:in std_logic;
            sreset,sa,rdd,ben:in std_logic;
            breset,wrd,clk,rst,dma_cs:in std_logic;
            spi_mosi:in std_logic;
            spi_miso:out std_logic;
            equ,cs,smequ:out std_logic;
            addr_port:out std_logic_vector(7 downto 0);
            data_port:in std_logic_vector(7 downto 0));
end staza;

architecture Behavioral of staza is
      signal sh:std_logic_vector(97 downto 0);
      signal i1,i2,i3,i4,index,barout,const:std_logic_vector(7 downto 0);
      signal baza,zbirout,ukupno:std_logic_vector(7 downto 0);
      signal addra,addrb,reg_data,brojac,brojb:std_logic_vector(7 downto 0);
      signal sm:std_logic_vector(1 downto 0);
      signal dek:std_logic_vector(3 downto 0);
      signal enb1,enb2,enb3,enb4:std_logic;

      -- barel pomerac
      component barrel is
      Port(
            index: in std_logic_vector(7 downto 0);
            sb: in std_logic_vector(1 downto 0);
            barout:out std_logic_vector(7 downto 0));
      end component;

      --brojac indeksa
      component brojac_i is
      Port(
            input: in std_logic_vector(7 downto 0);
            wr,clk,rst,en: in std_logic;
            output: out std_logic_vector(7 downto 0));
      end component;

begin
ii1: brojac_i port map (input=>sh(7 downto 0),
                        wr=>wrb,en=>enb1,clk=>clk,rst=>rst,output=>i1);
ii2: brojac_i port map (input=>sh(15 downto 8),
```

```
                        wr=>wrb,en=>enb2,clk=>clk,rst=>rst,output=>i2);
ii3: brojac_i port map (input=>sh(23 downto 16),
                        wr=>wrb,en=>enb3,clk=>clk,rst=>rst,output=>i3);
ii4: brojac_i port map (input=>sh(31 downto 24),
                        wr=>wrb,en=>enb4,clk=>clk,rst=>rst,output=>i4);

barel: barrel port map(index=>index,sb=>sh(97 downto 96),barout=>barout);

--multiplekser za izbor indeksa
index<=i1 when(sm="00") else
       i2 when(sm="01") else
       i3 when(sm="10") else
       i4;

--multiplekser za izbor konstanti
const<="00000001" when(sm="00") else
       "00000100" when(sm="01") else
       "00010000" when(sm="10") else
       "10000000";

--multiplekser za izbor bazne adrese
baza<=sh(39 downto 32) when(sm="00") else
      sh(47 downto 40) when(sm="01") else
      sh(55 downto 48) when(sm="10") else
      sh(63 downto 56);

--multiplekser za izbor adrese
addr_port<=addra when(sa='1') else addrb;

--multiplekser za izbor velicine bloka
brojb<=sh(71 downto 64) when(sm="00") else
       sh(79 downto 72) when(sm="01") else
       sh(87 downto 80) when(sm="10") else
       sh(95 downto 88);

--sabirac 1
zbirout<=std_logic_vector(unsigned(const)+unsigned(baza));

--sabirac 2
ukupno<=std_logic_vector(unsigned(barout)+unsigned(zbirout));
addrb<=ukupno;

--registar addra
process(clk)
begin
      if(clk 'event and clk='1') then
            if(sreset='1') then
                  addra<=(others=>'0');
            elsif(sreg2='1') then
                  addra<=std_logic_vector(unsigned(addra)+1);
            end if;
      end if;
```

```
end process;

--brojac prenetih bajtova
process(clk)
begin
	if(clk 'event and clk='1') then
		if(breset='1') then
			brojac<=(others=>'0');
		elsif(ben='1') then
			brojac<=std_logic_vector(unsigned(brojac)+1);
		end if;
	end if;
end process;

--komparator
equ<='1' when(unsigned(brojac)=unsigned(brojb)) else '0';

--bufer
process(clk)
begin
	if(clk 'event and clk='1') then
		if(wrd='1') then
			reg_data<=data_port;
		end if;
	end if;
end process;

--3state buffer
data_port<=reg_data when(rdd='1') else (others=>'Z');

--registar sh
process(spi_clk)
begin
	if(spi_clk 'event and spi_clk='1') then
		if(dma_cs='1') then
			sh<=spi_mosi&sh(97 downto 1);
		end if;
	end if;
end process;

--spi_miso
spi_miso<=sh(0);

--brojac sm 2-bit, broji kanale
process(clk)
begin
	if(clk 'event and clk='1') then
		if(smreset='1') then
			sm<=(others=>'0');
		elsif(smen='1') then
			sm<=std_logic_vector(unsigned(sm)+1);
		end if;
```

```
      end if;
end process;

smequ<='1' when (sm="11") else '0';

--dekoder 2u4 sm=>en
dek<="0000" when (enb='0') else
     "0001" when(sm="00") else
     "0010" when(sm="01") else
     "0100" when(sm="10") else
     "1000";

enb4<=dek(3);
enb3<=dek(2);
enb2<=dek(1);
enb1<=dek(0);

cs<=dma_cs;

end Behavioral;
```

Kontrolna jedinica: automat.vhd

```
----------------------------------------------------------------------------
-- Automat
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity automat is
      Port(
            clk,rst,equ,cs,smequ:in std_logic;
            wrb:out std_logic;
            sreg2,sreset,sa,rdd:out std_logic;
            ben,breset,wrd,rd,wr,ready,smen,smreset,enb:out std_logic);
end automat;

architecture Behavioral of automat is
      type state is (idle,puni,s,upis,citanje,pon);
      signal pr_state,nx_state:state;
begin
process(clk,rst)
begin
      if(rst='1') then
            pr_state<=idle;
      elsif(clk 'event and clk='1') then
            pr_state<=nx_state;
      end if;
end process;

--logika sledeceg stanja
process(smequ,equ,cs,pr_state)
begin
      wrb<='0';
      sreg2<='0';sreset<='0';sa<='0';rdd<='0';ben<='0';breset<='0';
      wrd<='0';rd<='0';wr<='0';ready<='0';smen<='0';smreset<='0';enb<='0';
      nx_state<=pr_state;
      case pr_state is
      when idle=>
            ready<='1';
            if(cs='1') then
                  nx_state<=puni;
            end if;
      when puni=>
            if(cs='0') then
                  sreset<='1';
                  breset<='1';
                  smreset<='1';
                  wrb<='1';
                  nx_state<=s;
            end if;
      when s=>
```

```
            rd<='1';
            sa<='1';
            nx_state<=citanje;
      when citanje=>
            rd<='1';
            wrd<='1';
            sa<='1';
            nx_state<=upis;
      when upis=>
            sreg2<='1';
            wr<='1';
            rdd<='1';
            nx_state<=pon;
      when pon=>
            wr<='1';
            rdd<='1';
            ben<='1';
            enb<='1';
            if(equ='1') then
                  breset<='1';
                  smen<='1';
                  if(smequ='1') then
                        nx_state<=idle;
                  else
                  nx_state<=s;
                  end if;
            else
                  nx_state<=s;
            end if;
      end case;
end process;

end Behavioral;
```

DMA: dma.vhd

```
----------------------------------------------------------------------------
-- Spajanje kontrolne jedinice i staze podataka
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DMA is
      Port(
            clk,rst,dma_cs,spi_clk,spi_mosi:in std_logic;
            data_port: in std_logic_vector(7 downto 0);
            addr_port:out std_logic_vector(7 downto 0);
            spi_miso,ready,rd,wr:out std_logic);
end DMA;

architecture Behavioral of DMA is
      --instanciranje staze podataka dma
      component staza is
      Port(
            wrb,spi_clk,enb:in std_logic;
            smen,smreset:in std_logic;
            sreg2:in std_logic;
            sreset,sa,rdd,ben:in std_logic;
            breset,wrd,clk,rst,dma_cs:in std_logic;
            spi_mosi:in std_logic;
            spi_miso:out std_logic;
            equ,cs,smequ:out std_logic;
            addr_port:out std_logic_vector(7 downto 0);
            data_port:in std_logic_vector(7 downto 0));
      end component;

      --instanciranje automata
      component automat is
      Port(
            clk,rst,equ,cs,smequ:in std_logic;
            wrb:out std_logic;
            sreg2,sreset,sa,rdd:out std_logic;
            ben,breset,wrd,rd,wr,ready,smen,smreset,enb:out std_logic);
      end component;

      signal enb,wrb,sreg2,sreset,sa,rdd,ben,breset,wrd:std_logic;
      signal smen,smreset,equ,smequ,cs:std_logic;
begin
--staza
stazica:staza port map(wrb=>wrb,spi_clk=>spi_clk,enb=>enb,smen=>smen,
                       smreset=>smreset,sa=>sa,rdd=>rdd,
                       ben=>ben,breset=>breset,wrd=>wrd,
                       sreg2=>sreg2,sreset=>sreset,clk=>clk,
                       rst=>rst,equ=>equ,cs=>cs,smequ=>smequ,
                       spi_mosi=>spi_mosi,spi_miso=>spi_miso,
```

```
                        addr_port=>addr_port,
                        data_port=>data_port,dma_cs=>dma_cs);

auto:automat port map(clk=>clk,rst=>rst,equ=>equ,cs=>cs,smequ=>smequ,
                      enb=>enb,wrb=>wrb,sreg2=>sreg2,sreset=>sreset,
                      sa=>sa,rdd=>rdd,ben=>ben,breset=>breset,wrd=>wrd,
                      rd=>rd,wr=>wr,smen=>smen,
                      smreset=>smreset,ready=>ready);

end Behavioral;
```

Literatura

Mile Stojčev-RISC,CISC i DSP procesori
Mile Stojčev-Embeded sistemi
Goran Lj. Đorđević- Arhitekture mikrosistema
K.C.Chang-Digital Systems Design with VHDL and Synthesis An Integrated Approach
Douglas L. Perry-VHDL:Programming by Exsample

www.altera.com
www.xilinx.com
www.intel.com