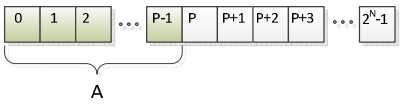
RTL projektovanje – projekat

1. Specifikacija projekta

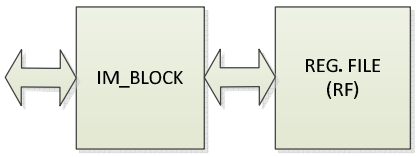
Inteligentna memorija (IM) je memorijska komponenta koja osim standardnih RAM-funkcija upisa i čitanja poseduje mogućnost izvesne obrade memorisanog sadržaja. Tako, IM je u mogućnosti da autonomno izvrši sortiranje upisanih podataka, pretragu po zadatom kriterijumu, umetanje i izbacivanje podataka i sl.

Kapacitet IM iznosi 2^NxB, tj. 2^N B-bitnih memorijskih lokacija. Smatraćemo da je u IM upisan niz A od P elemenata koji zauzima prvih P lokacija raspoloživog memorijskog prostora ($P \leq 2^N$-1) - Sl. 1. Dužina niza A se menja upisom ili izbacivanjem podataka u/iz niza. Na Sl. 1, ″obojena″ polja predstavljaju memorijske lokacije zauzete nizom A.



Sl. 1

IM se sastoji iz dva glavna dela: RAM u obliku registarskog fajla i IM_BLOCK koji realizuje specifične IM operacije (Sl. 2). Projektni zadatak se odnosi na projektovanje IM bloka koji u sprezi sa registarskim fajlom obavlja jednu konkretnu IM-operaciju. Projektni zadaci se prevashodno odnose na projektovanje IM bloka, dok registarski fajl treba koristiti kao gotovu komponentu. (VHDL opis registarskog fajla koji je dat u sledećoj sekciji treba uključiti u projekat i koristiti kao komponentu).

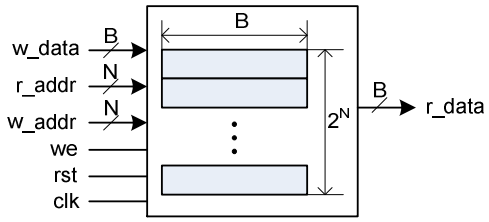


Sl. 2

1.1 Registarski fajl

Registarski fajl je skup registara s jednim ulaznim i jednim ili više izlaznih portova. Ispod je dat VHDL model registarskog fajla s jednim ulaznim i jednim izlaznim portom, sa sinhronim upisom i asinhronim čitanjem, koji će biti korišćen kao memorijska komponenta u IM sistemu. Blok dijagram ovog registarskog fajla prikazan je na Sl. 3. Adresa upisa, *w_addr*, određuje registar u koji će biti upisan podatak prisutan na ulaznom portu *w_data*, dok adresa čitanja, *r_addr*, određuje registar čiji se sadržaj prenosi na izlazni port *r_data*. Upis je iniciran rastućom ivicom taktnog signala *clk*. Upis podataka je sinhron, sa dozvolom upisa *we*. Čitanje je asinhrono, tj. ne postoji eksplicitna dozvola čitanja i nije sinhronizovano sa taktom (svaka

promena adrese *r_addr* inicira čitanje iz adresiranog registra). Dati VHDL model predstavlja parametrizovan opis 2^NxB registarskog fajla.



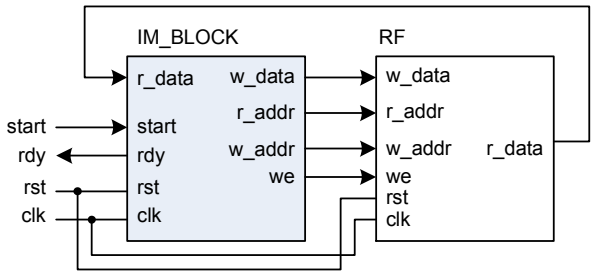
Sl. 3

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
--------------------------------------------------------------------------------
ENTITY reg_file IS
  GENERIC(N : INTEGER;   -- broj adresnih bita
          B : INTEGER);  -- broj bita u reci
  PORT(clk, rst : IN STD_LOGIC;
       we : IN STD_LOGIC;
       w_addr, r_addr : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
       w_data : IN STD_LOGIC_VECTOR(B-1 DOWNTO 0);
       r_data : OUT STD_LOGIC_VECTOR(B-1 DOWNTO 0));
END reg_file;
--------------------------------------------------------------------------------
ARCHITECTURE arch OF reg_file IS
  TYPE reg_file_type IS ARRAY(2**N-1 DOWNTO 0) OF
       STD_LOGIC_VECTOR(B-1 DOWNTO 0);
  SIGNAL array_reg : reg_file_type;
BEGIN
  -- upis
  PROCESS(clk, rst)
  BEGIN
    IF(rst = '1') THEN
      array_reg <= (OTHERS => (OTHERS => '0'));
    ELSIF(clk'event AND clk = '1') THEN
      IF(we = '1') THEN
        array_reg(TO_INTEGER(UNSIGNED(w_addr))) <= w_data;
      END IF;
    END IF;
  END PROCESS;
  -- port za citanje
  r_data <= array_reg(TO_INTEGER(UNSIGNED(r_addr)));
 END arch;
```

1.2 Projektni zadaci

1. **Zadatak:** Sortiranje niza – klasičan algoritam sortiranja

   Projektovati IM blok za sortiranje niza *A* neoznačenih celih brojeva. Blok dijagram sistema prikazan je na Sl. 4(a), a algoritam po kome radi IM blok na Sl. 4(b). Reč je o klasičnom algoritmu sortiranja po kome se u svaku poziciju niza, redom, s leva udesno, prebacuje najmanji element iz preostalog, nesortiranog dela niza.



(a)

```
for(i=0;i<P-1;i++){
   min = A[i];
   k = i;
   for(j=i;j<P;j++){
      if(A[j]<min){
         min = A[j];
         k = j;
      }
   }
   A[i]↔A[k]; //zamena
}
```

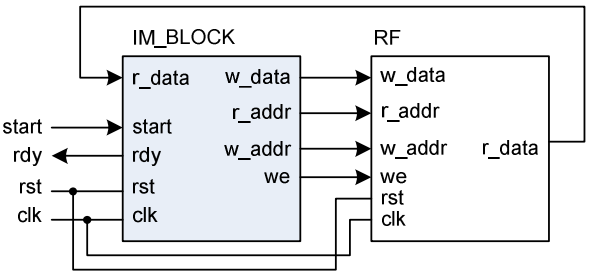
(b)

(c)

Sl. 4

2. Zadatak: Sortiranje niza – *BubbleSort* algoritam sortiranja

Projektovati IM blok za sortiranje niza *A* neoznačenih celih brojeva. Blok dijagram sistema prikazan je na Sl. 5(a), a algoritam po kome radi IM blok na Sl. 5(b). Reč je o tzv. *bubble sort* algoritmu sortiranja. U unutrašnjoj petlji ovog algoritma prolazi se unazad kroz niz, s desna ulevo sve do pozicije indeksa spoljašnje petlje, *i*, i pri tom vrši zamena mesta susednih elemenata tako da veći dođe desno, a manji levo. Na taj način, po završetku unutrašnje petlje, na poziciju indeksa *i* dolazi najmanji element u pređenom delu niza. Budući da *i* kreće od nule i da se sukcesivno pomera ka kraju niza, konačni ishod algoritma je sortirani niz *A*.



(a)

```
for(i=0;i<P-1;i++){
  for(j=P-1;j>i;j--)
    if(A[j]<A[j-1]){
      A[j]↔A[j-1]; //zamena
    }
}
```

(b)

(c)

Sl. 5

3. Zadatak: Pretraga u sortiranom nizu

Projektovati IM blok za brzu proveru prisutnosti zadatog elementa u niz (Sl. 6(a)). Korisnik postavlja traženu vrednost, *D*, na ulazni port *din* i startuje pretragu aktiviranjem signala *start*. Nakon obavljene pretrage, sistem aktivira signal *rdy*, a rezultat pretrage

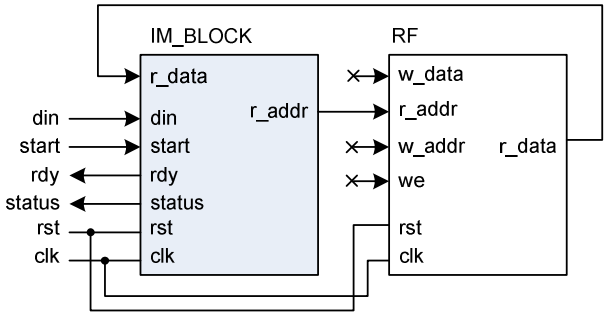
postavlja na izlazu *status* (*status*=0 - element nije pronađen; *status*=1 – element je pronađen).

Radi ubrzanja pretrage, niz *A* je sortiran u rastućem redosledu sa elementom čija je brojna vrednost najmanja na lokaciji 0. Rad sistema je zasnovan na algoritmu pretrage *BinarySearch*, koji je u obliku pseudo koda prikazan na Sl. 6(b).

U svakoj iteraciji algoritma postoji interval elemenata u nizu, omeđen donjom granicom, *x* i gornjom granicom, *y* koji sadrži traženi element (pod pretpostavkom da traženi element postoji u nizu). Algoritam ispituje središnji elemenat intervala (tj. element niza s indeksom $i=(x+y)/2$). Ako je A[*i*]=*D*, element je pronađen i algoritam završava rad sa *status* =1. Inače, ako je A[*i*] različito od *D*, interval se skraćuje tako što se pomera *x* ili *y*, u zavisnosti od toga da li je A[*i*] manje ili veće od *D*. Ako je *A*[*i*]<D, pomera se *x* tako što dobija vrednost za 1 veću od indeksa ispitivanog elementa niza. U suprotnom, ako je *A*[*i*]>*D*, pomera se gornja granica, *y*, tako što dobija vrednost za jedan manju od *i*.

Na primer, neka važi: *A* = (1, 5, 16, 29, 38, 44, 58, 59, 66) i neka je traženi element *D*=44 (Sl. 6(c)). S obzirom da je dužina niza *P*=9, na početku važi *x*=0 i *y*=8. U prvoj iteraciji, tražena vrednost se poredi sa elementom niza sa indeksom $i = (x+y)/2 = 4$, tj. sa *A*[4] = 38. Pošto je 38 < 44, donja granica intervala *x* dobija vrednost, $x=i+1=5$. U drugoj iteraciji, ispituje se element sa indeksom $i = (x+y)/2 = (5 + 8)/2 = 6$, tj. *A*[6] = 58. Pošto je sada 58 > 44, pomera se gornja granica intervala *y*, koja dobija vrednost, tj. $y = i-1 = 5$. Treća iteracija ispituje *A*[(5+5)/2]=*A*[5]. Pošto važi *A*[5]=44, pretraga je uspešno završena i sistem završava rad sa *status* =1.

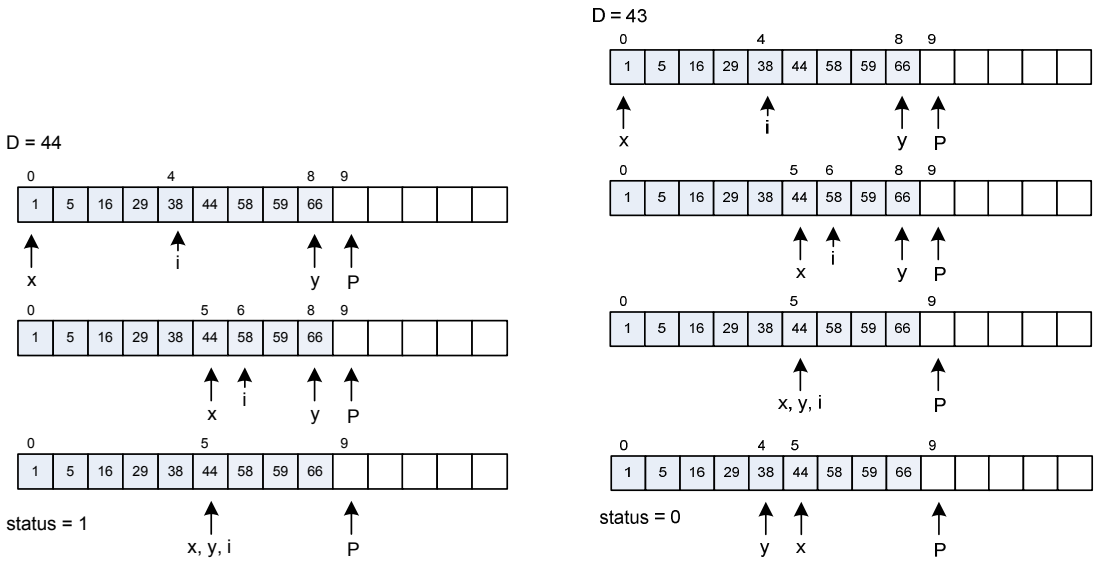
Da je umesto 44 tražena šifra bila *D*=43. algoritam bi prošao i kroz četvrtu iteraciju, u kojoj bi gornja granica dobila vrednost $y = i - 1 = 4$ (Sl. 6(d)). Time bi uslov *while* petlje, $x <= y$, postao netačan i algoritam bi završio rad sa *status =0*.



(a)

```
status = 0;
x = 0; y= P-1;
while(x <= y && !status){
   i = (x + y)/2;
   if(A[i] == D){
      status = 1;
   }else if(A[i] < D){
      x = i + 1;
   }else{
      y = i - 1;
   }
}
```

(b)

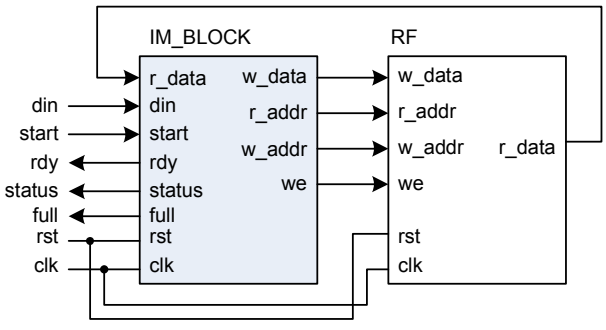


(c) (d)

Sl. 6

4. Zadatak: Umetanje elementa u sortirani niz

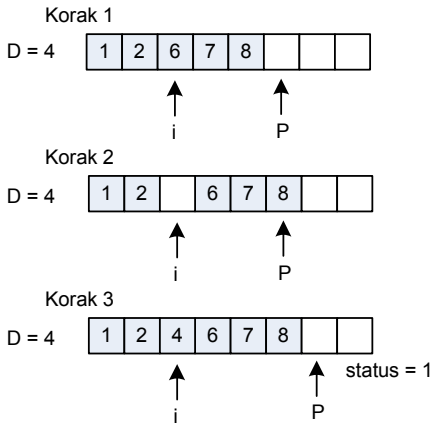
Projektovati IM blok za umetanje novog elementa *D* u sortirani niz *A* (Sl. 7(a)). Algoritam po kome IM blok treba da radi dat je na Sl. 7(b). U prvom delu algoritma (linije 1-4), traži se pozicija na koju *D* treba da bude upisano. To se radi tako što se prolazi kroz niz *A*, od početka pa sve do prvog elementa koji je jednak ili veći od *D* (vidi Sl. 7(c), korak 1). Ako je pronađeni element jednak *D*, nema potrebe za upisom nove vrednosti i algoritam završava rad sa *status*=0 (linije 5-6). Inače, ako je u nizu *A* pronađen element koji je veći od *D*, u dugom delu algoritma (linije 8-12) pravi se prazan prostor u nizu *A* za umetanje elementa *D* (vidi Sl. 7(c), korak 2). Konačno, u trećem delu algoritma, *D* se upisuje na ispražnjenu poziciju i *P* se povećava za 1 (vidi Sl. 7(c), korak 3).



(a)

```
1  i = 0;
2  while(i < P && A[i]<D){
3    i = i + 1;
4  }
5  if(i < P && A[i]=D){
6   status = 0;
7  else {
8    j = P;
9    while(j > i){
10     A[j] = A[j - 1];
11     j = j - 1;
12   }
13   A[i] = D
14   P = P + 1;
15   status = 1;
16 }
```

(b)

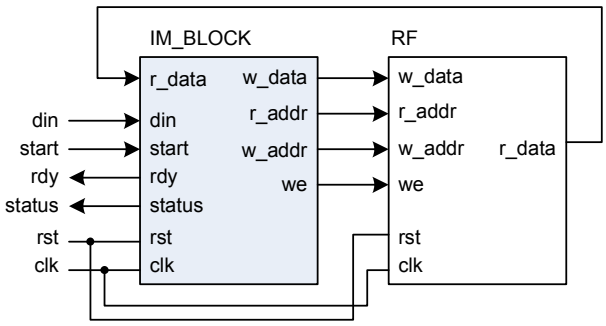


(c)

Sl. 7

5. Zadatak: Izbacivanje elementa iz sortiranog niza

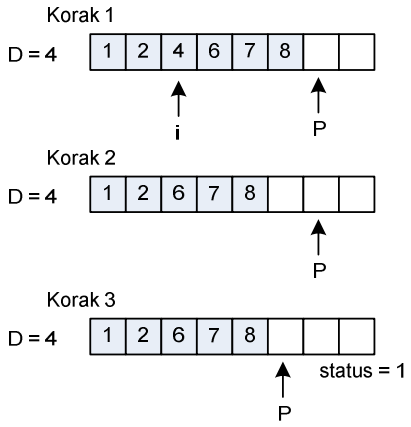
Projektovati IM blok za izbacivanje elementa *D* iz sortiranog niz *A* (Sl. 8(a)). Algoritam po kome IM blok treba da radi dat je na Sl. 8(b). U prvom delu algoritma (linije 1-4), traži se pozicija elementa *D* u nizu *A*. To se radi tako što se prolazi kroz niz *A*, od početka pa sve do prvog elementa koji je jednak ili veći od *D* (vidi Sl. 8(c), korak 1). Ako se stiglo do kraja niza ili do elementa koji je veći od *D*, to znači da u nizu *A* element *D* ne postoji, pa algoritam završava rad sa *status*=0 (linije 5-6). Inače, ako je u nizu *A* pronađen element *D*, u dugom delu algoritma (linije 8-11) element *D* se istiskuje iz niza tako što se svi elementi koji slede pomeraju za jednu poziciju udesno. (vidi Sl. 8(c), korak 2). Konačno, u trećem delu algoritma, *P* se smanjuje za 1 (vidi Sl. 8(c), korak 3).



(a)

```
1  i = 0;
2  while(i<P && A[i]<D){
3    i = i + 1;
4  }
5  if(i=P || (i<P && A[i]≠D)){
6    status = 0;
7  }else {
8     while(i < P-1){
9       A[i} = A[i+1];
10      i = i + 1;
11    }
12    P = P – 1;
13    status = 1;
14 }
```

(b)

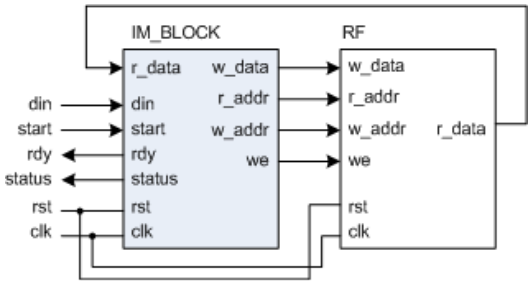


(c)

Sl. 8

6. Zadatak: Izbacivanje elementa iz nesortiranog niza

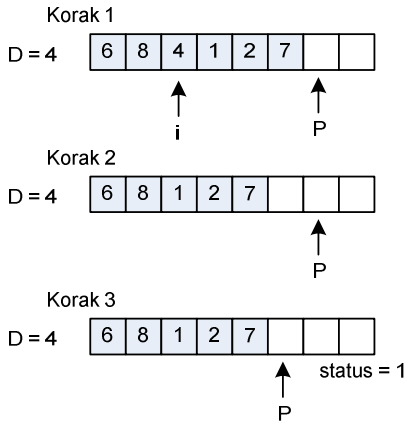
Projektovati IM blok za izbacivanje elementa *D* iz nesortiranog niz *A* (Sl. 9(a)). Algoritam po kome IM blok treba da radi dat je na Sl. 9(b). U prvom delu algoritma (linije 1-4), traži se pozicija elementa *D* u nizu *A*. To se radi tako što se prolazi kroz niz *A*, od početka pa sve do elementa koji je jednak *D* (vidi Sl. 9(c), korak 1). Ako se stiglo do kraja niza, a da *D* nije nađeno, algoritam završava rad sa *status*=0 (linije 5-6). Inače, ako je u nizu *A* pronađen element *D*, u dugom delu algoritma (linije 8-11) element *D* se istiskuje iz niza tako što se svi elementi koji slede pomeraju za jednu poziciju udesno. (vidi Sl. 9(c), korak 2). Konačno, u trećem delu algoritma, *P* se smanjuje za 1 (vidi Sl. 9(c), korak 3).



(a)

```
1  i = 0;
2  while(i<P && A[i]≠D){
3    i = i + 1;
4  }
5  if(i=P){
6    status = 0;
7  }else {
8     while(i < P-1){
9       A[i} = A[i+1];
10      i = i + 1;
11    }
12    P = P – 1;
13    status = 1;
14 }
```

(b)



(c)

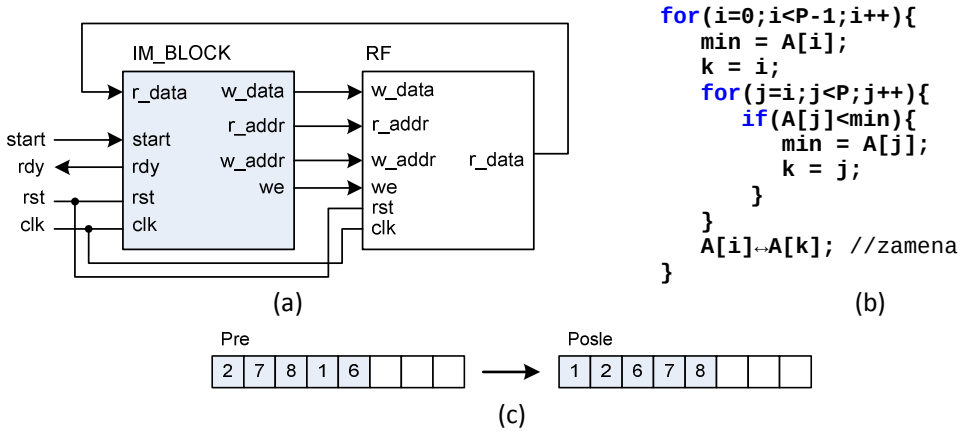
Sl. 9

6.1.1 Napomene i preporuke

1. IM blokovi (praktično svi, osim zadatka 4) manipulišu nad podacima koji su prethodno, na neki način, upisani u registarski fajl. To znači da u testbeču za takve IM blokove treba obezbediti upis inicijalnog sadržaja u registarski fajl (videti 6.2).
2. Strukturni način opisa treba koristiti u modulima koji služe za povezivanje komponenata (*im_block.vhd* i *im.vhd*), dok VHDL module staze podataka i upravljačke jedinice treba opisati na funkcionalnim novu. To se prevashodno odnosi na stazu podataka koju treba opisati jednim VHDL modulom. Staze podataka za sve projektne zadatke su relativno jednostavne i nema potrebe da se dalje, strukturno razlažu na jednostavnije komponente.
3. Primarni projektantski zahtev je optimizacija performansi (brzine rada), a sekundarni hardverska složenost. To praktično znači da prilikom optimizacije ASMD dijagrama treba težiti smanjenju broja stanja (taktnih ciklusa), a pre svega broja stanja u petljama. S druge strane, prilikom projektovanja staze podataka treba sprovesti optimizaciju primenom tehnike deobe operatora/funkcija sa ciljem da se minimizira broj operatora, odnosno broj funkcionalnih jedinica. Na primer, ako se ista operacija javlja u dva različita stanja ASMD dijagrama, racionalnije je umesto dve identične funkcionalne jedinice od kojih se svaka koristi u jednom stanju, upotrebiti jednu koja će se koristiti u oba stanja (moguće sa različitim ulaznim operandima).

6.2 Sortiranje niza – klasičan algoritam sortiranja

U ovoj sekciji biće predstavljeno kompletno rešenje zadatka 1 – klasičan algoritam sortiranja (Sl. 10). Tok projektovanja obuhvata sledeće korake: 1) prilagođenje interfejsa; 2) prilagođenje algoritma, 3) transformacija algoritma u ASMD dijagram, 4) optimizacija ASMD dijagrama, 5) projektovanje staze podataka, 6) konverzija ASMD dijagrama u ASM dijagram upravljačke jedinice, 7) kreiranje VHDL opisa, i 8) simulacija.

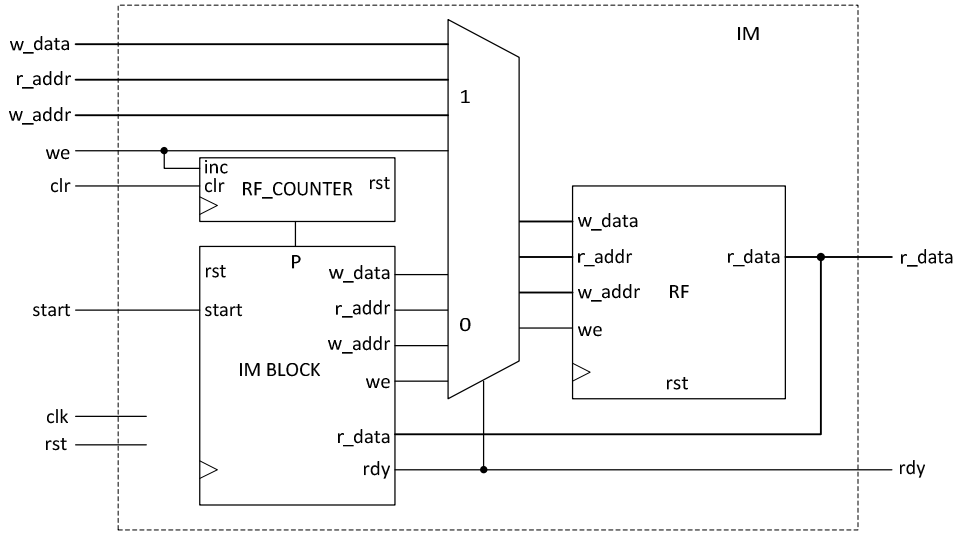


Sl. 10

6.2.1 Prilagođenje interfejsa

Blok dijagram sa Sl. 10(a) pokazuje spregu IM bloka i registarskog fajla. Međutim, kompletan interfejs bi trebalo da obuhvati i mogućnost pristupa registarskom fajlu mimo IM bloka (radi inicijalnog upisa niza). Takođe, IM bloku je potrebno dostaviti informaciju o broju elemenata u registarskom fajlu (parametar *P*).

Na Sl. 11 je prikazan blok dijagram kompletnog IM sistema. Za vreme dok je IM blok u pasivnom stanju (*rdy* = 1), registarskom fajlu je moguće pristupati preko eksternih portova (radi upisa inicijalnog sadržaja ili radi čitanja upisanog sadržaja); za vreme dok je IM blok u operativnom režimu (obavlja sortiranje, *rdy*=0), registarskom fajlu može isključivo da pristupa IM blok. Blok RF_COUNTER je brojač koji prebrojava podatke koji se preko eksternih portova upisuju u registarski fajl. Sadržaj ovog brojača odgovara vrednosti parametra *P*. Signal *clr* služi za sinhrono resetovanje RF brojača – efekat je isti ko pražnjenje registarskog fajla.



Sl. 11

6.2.2 Prilagođenje algoritma

Algoritam, prilagođen transformaciji u ASMD dijagram (for petlja je zamenjena *if – goto* naredbama):

```
        i=0;
lab1:   if(i<P-1){
          min = A[i];
          k = i;
          j = i;
lab2:     if (j<P){
             if(A[j]<min){
                min = A[j];
                k = j;
             }
             j = j+1;
             goto lab2;
          }
          A[k] = A[i];
          A[i] = min;
          i = i+1;
          goto lab1;
      }
```

6.2.3 Konverzija algoritma u ASMD dijagram

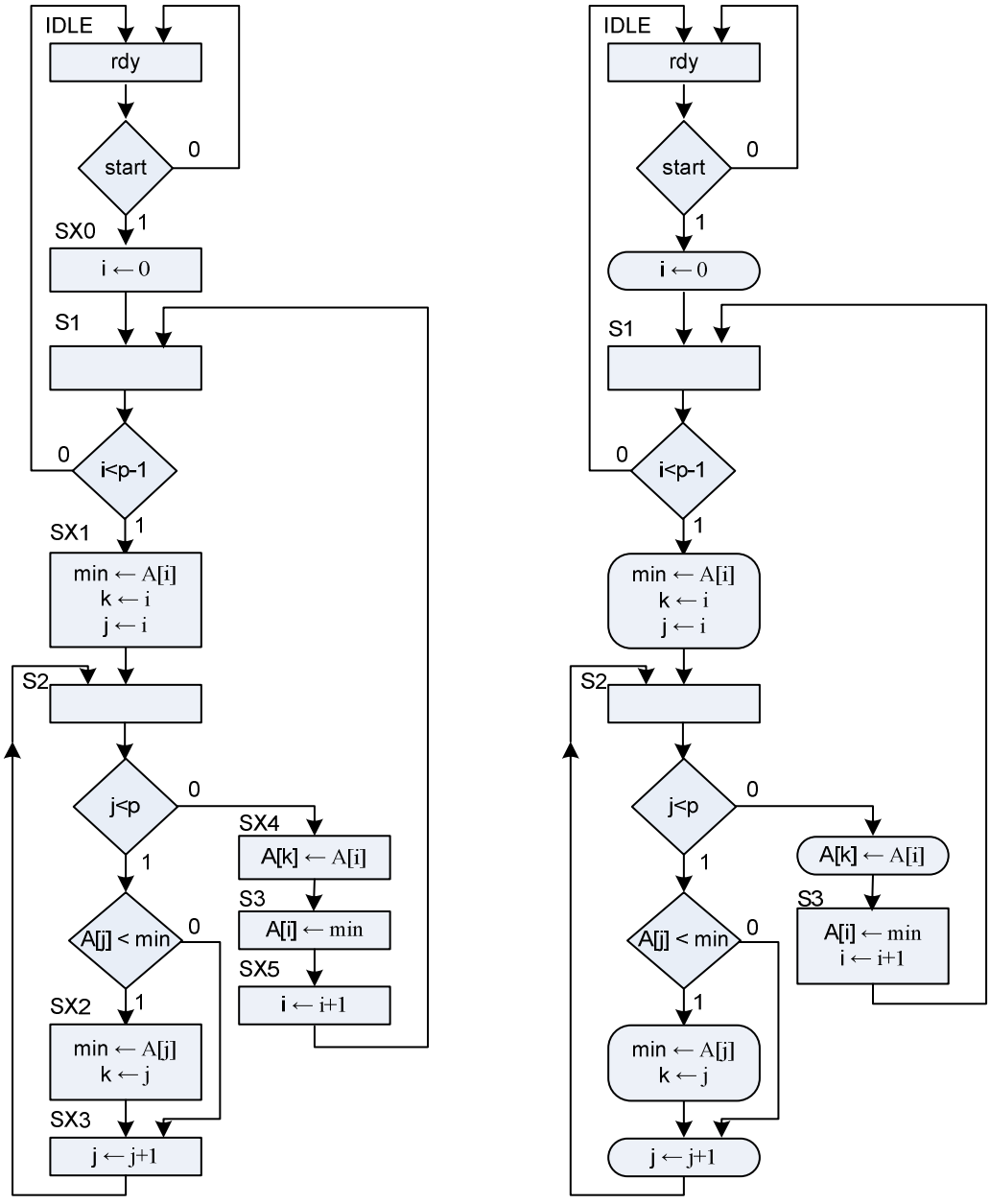
Na osnovu prilagođenog algoritma, dobija se ASMD dijagram sistema za sortiranje prikazan na Sl. 12(a). U polaznom ASMD algoritmu neke operacije su već objedinjene u ista stanja, zato što se radi o očigledno nezavisnim operacijama.

6.2.4 Optimizacija ASMD dijagrama

Optimizovan ASMD dijagram je prikazan na Sl. 12(b). Optimizacije su izvršene na sledeći način:

- Stanje SX0 je pripojeno stanju IDLE što znači da će registar *i* biti resetovan u istom taktnom ciklusu kad signal *start* postane jednak 1, a ne u sledećem.
- Prazno stanje S1 se ne može eliminisati jer bi se time ispitivanje i ≤ p-1 pripojilo stanjima IDLE i S7 u kojima se vrši upis u *i*.
- Stanje SX1 se pripaja stanju S1 zato što operacije iz ovog stanja zavise isključivo od sadržaja registra *i*, a u registar se ne vrši upis u stanju S1.
- Stanje SX2 se pripaja stanju S2 jer operacije u SX2 zavise samo od *j*, a u registar *j* se ništa ne upisuje u stanju S2.
- Stanje SX3 se takođe pripaja stanju S2. Ovo je korektno bez obzira što će se tako operacija inkrementiranja registra *j* preklopiti u vremenu sa operacijama iz starog stanja SX2 gde se *j* koristi kao ulazni operand. Korektno je zato što se u operacijama iz SX2 koristi aktuelni sadržaj registra *j*, a inkrementirana vrednost će biti upisana u *j* tek u momentu završetka tekućeg taktnog ciklusa.
- Stanje SX4 pripojeno je stanju S2. Naredba registarskog prenosa iz SX4 ima dva ulazna operanda, *i* i *k*, *i* je adresa čitanja, a *k* adresa upisa u niz A. Spajanje je dopušteno jer: Registar *i* se ne menja u stanju S2. U registar *k* se upisuje u stanju S2 (nakon pripajanja SX2 stanju S2), ali pošto su operacije iz SX4 I SX2 uzajamno isključive, operacija iz SX6 će uvek koristiti ažurnu vrednost registra *k*.

- Stanje S3 se ne može pripojiti stanju S2, jer bi se time spojile u isto stanje dve operacije upisa u A (dopuštena je najviše jedna – takav je registarski fajl).
- Konačno, operacija inkrementiranja iz stanja SX5 može da pređe u S3 iako se u S3 sadržaj registra *i* koristi kao ulazni operand jer će upis inkrementirane vrednosti u *i* biti obavljen na kraju taktnog ciklusa



Sl. 12

6.2.5 Projektovanje staze podataka

Staza podataka se projektuje na osnovu optimizovanog ASMD dijagrama. Čitanje i upis u niz *A*, sa stanovišta hardvera, treba tumačiti na sledeći način:

- $A[x] \leftarrow y$ znači da na port *w_addr* treba postaviti *x*, a na port *w_data* postaviti *y* (uz *we*=1)
- $y \leftarrow A[x]$ znači da na port *r_addr* treba postaviti *x*. Pročitani podatak iz registarskog fajla je dostupan na portu *r_data* – treba ga sprovesti do registra *y* (uz aktiviranje signala dozvole upisa u *y*)
- $A[x] \leftarrow A[y]$ – znači da se na *r_addr* postavlja *y*, a na *w_addr* se postavlja *x*. Pročitani podatak iz registarskog fajla, koji je dostupan na portu *r_data* treba sprovesti na port *w_data* (uz *we*=1).

Registarske komponente: Prilikom izbora registarskih komponenti treba pratiti promenljive, tj. naredbe registarskog prenosa u kojima se vrši upis u promenljivu (registar):

- Promenljiva *i* – U stanju IDLE se resetuje, a u stanju S3 inkrementira. Za registar *i* biramo brojač sa sinhronim resetom.
- Promenljiva *j* – u stanju S1 u *j* se upisuje *i* a u stanju S2 *j* se inkrementira. Može se realizovati kao brojač sa paralelnim upisom.
- Promenljiva *k* – u stanju S1 se upisuje *i*, a u stanju S2 *j*, tako da biramo prihvatni registar sa dozvolom upisa.
- Promenljiva *min* – u stanju S1 se upisuje $A[i]$, tj. izlaz memorije, u stanju S2 $A[j]$, što je takođe izlaz memorije, tako da ovde biramo prihvatni registar sa dozvolom upisa, na čiji ulaz se dovodi izlaz memorije.
- Specifikacija registarskih komponenti je data u sledećoj tabeli:

Oznaka	**Tip**	**Promenljiva**	**Upravljački signali**	**Naredbe registarskog prenosa**
regi	Brojač sa sinhronim resetom	*i*	*ce* – dozvola brojanja *rst* – sinhrono resetovanje	$i \leftarrow 0$ (IDLE) $i \leftarrow i + 1$ (S3)
regj	Brojač sa paralelnim upisom	*j*	*ce* – dozvola brojanja *ld* – dozvola paralelnog upisa	$j \leftarrow i$ (S1) $j \leftarrow j + 1$ (S2)
regk	Registar sa dozvolom upisa	*k*	*ld* – dozvola upisa	$k \leftarrow i$ (S1) $k \leftarrow j$ (S2)
regmin	Registar sa dozvolom upisa	*min*	*ld* – dozvola upisa	*min* ← A[*i*] (S1) *min* ← A[*j*] (S2)

Operatori: U ASMD dijagramu se mogu identifikovati tri operatora poređenja (dva u stanju *S*2 i jedan u stanju *S*1) i operator dekrementiranja (u stanju *S*1). (Operatori inkrementiranja ($i + 1$ i $j+1$) obuhvaćeni su brojačima *regi* i *regj*.)

Za svaki operator treba ispitati mogućnost deobe (deoba operatora). U datom primeru, deoba je moguća, jer se operator poređenja *N*-to bitnih brojeva nalazi u dva različita stanja ($i<p-1$ u stanju *S*1 i $j<p$ u stanju *S*2). To znači da je moguće koristiti jedan komparator, sa dodatim multiplekserima za izbor operanada. Dalja deoba operatora nije moguće, jer se operator poređenja *B*-to bitnih brojeva ($A[j]<min$ u stanju *S*2) i operator dekrementiranja ($p-1$ u stanju S1) ionako koristi samo jedanput. To znači da su potrebna dva komparatora < i jedan dekrementer.

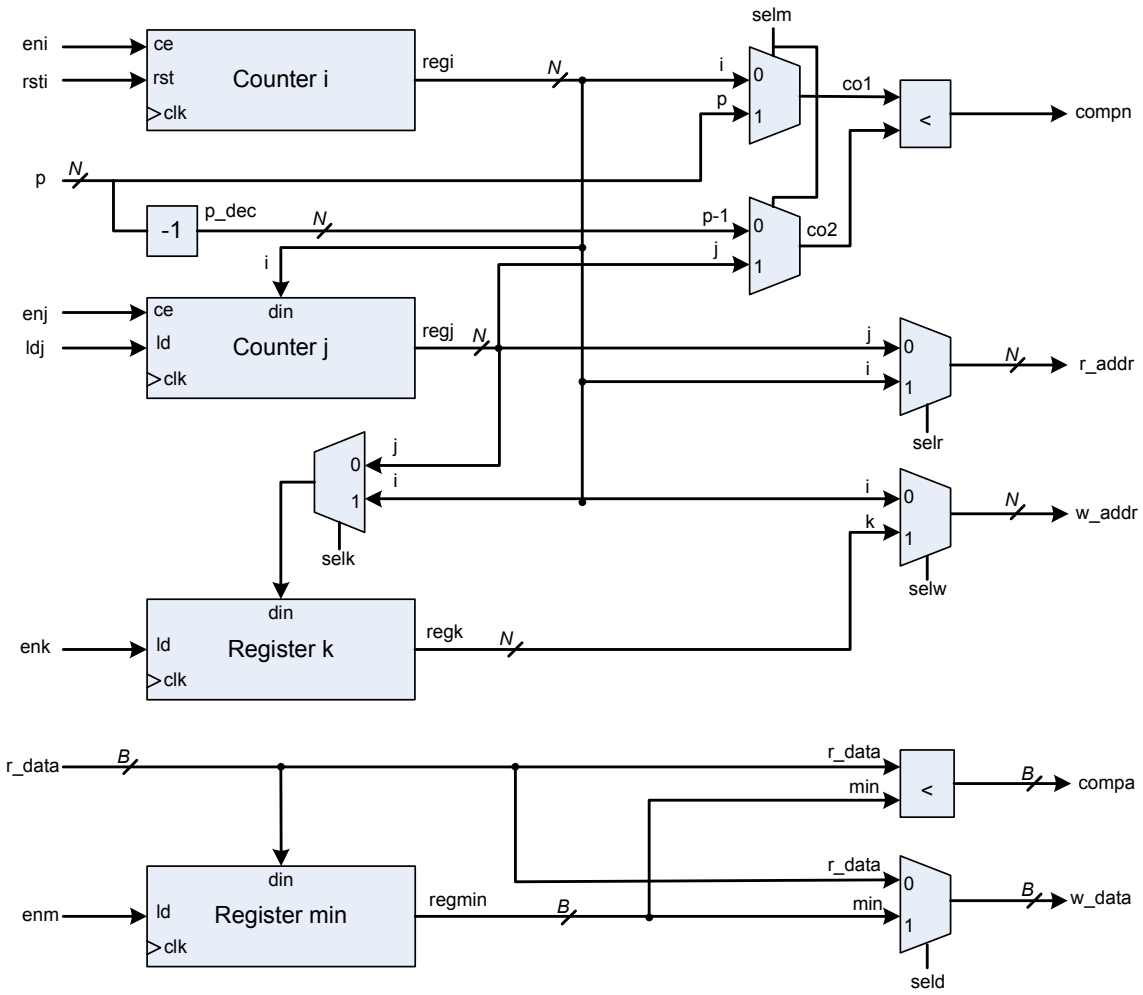
Aritmetičke kombinacione komponente:

Oznaka	**Tip**	**Operacija**	**Naredbe registarskog prenosa**
incr	Dekrementer	+1	$i < p$ - 1 (S1)
cmp1	Komparator manje od	<	$i < p$ - 1 (S1) $j < p$ (S2)
cmp2	Komparator manje od	<	A[i] < *min* (S2)

Povezivanje: Za svako odredište podataka (ulazi u registre sa paralelnim upisom i ulazi registarskog fajla) treba identifikovati izvore.

- Brojač *j*: Jedina operacija upisa u *j* je u stanju *S*1. Multiplekser nije potreban, direktna veza.
- Registar *k*: U stanju *S*1, u registar *k se* upisuje *i*; u stanju S2 u registar *k* se upisuje *j*. Na ulaz registra *k* je potrebno postaviti multiplekser 2-u-1.
- Registar *min*: Jedini upisu u registar *min* je u stanju *S*1, tj. onda kad se u ovaj registar upisuje podatak pročitan iz registarskog fajla. Dakle, *r_data* na *din* registra *min*.
- Port *r_addr*: Kao adresa za čitanje registarskog fajla koristi se *i* u stanju *S*1 i *i* ili *j* u stanju *S*2. Dakle, multiplekser 2-u-1.
- Port *w_addr*: Kao adresa za upis u registarski fajl koristi se *k* u stanju S2 i *i* u stanju S3. Na ovaj ulaz registarskog fajla treba postaviti multiplekser 2-u-1.
- Port *w_data*: U registarski fajl se upisuje podatak sa *r_data* u stanju *S*2 i sadržaj registra *min* u stanju S3. Na ovom ulazu je multiplekser 2-u-1.

Na osnovu prethodnih razmatranja pristupa se crtanju staze podataka, Sl. 13.



Sl. 13

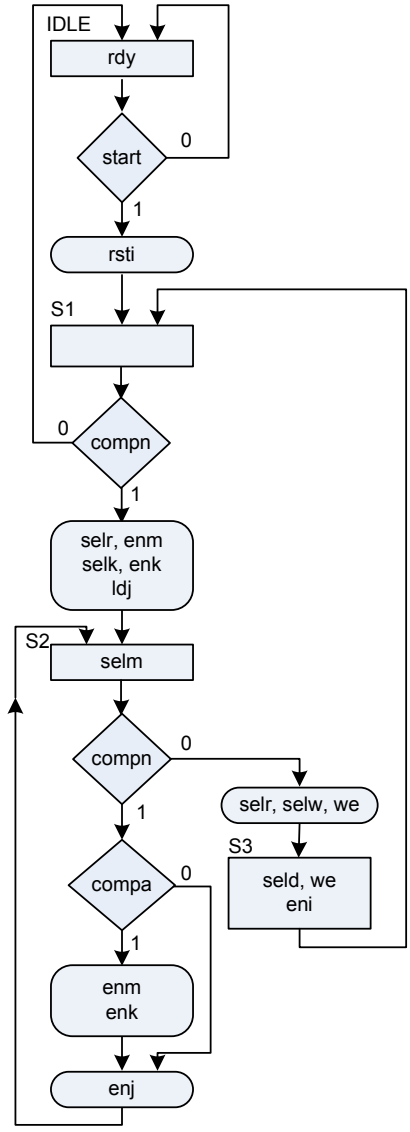
6.2.6 Konverzija ASMD u ASM upravljačke jedinice

Sl. 14 prikazuje ASM dijagram odgovarajuće upravljačke jedinice. Aktivnosti, po stanjima, su sledeće:

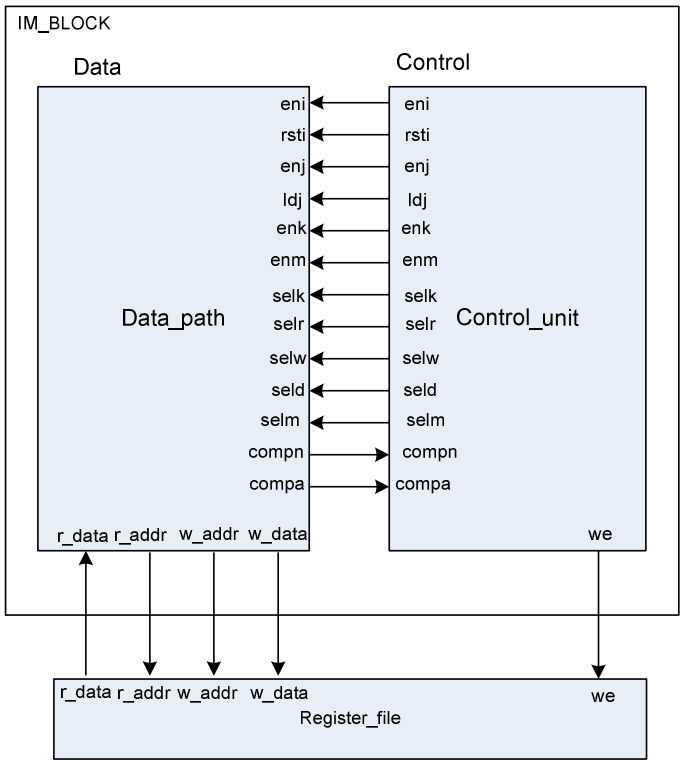
- **IDLE**: u ovom stanju se čeka početak rada (signal *start*). Ukoliko se desio, treba resetovati registar *i*, postavljanjem vrednosti *rsti*='1'.
- **S1**: provera da li je $i < p\text{-}1$ se dobija na izlazu komparatora *compn*, pod uslovom da je *selm* = '0'. Ukoliko jeste, izvršavaju se naredbe: $min \leftarrow A[i]$ (*selr*='1' i *enm*='1'); $k \leftarrow i$ (*selk*='1' i *enk*='1'); $j \leftarrow i$ (*ldj*='1');
- **S2**: Da bi se proverilo da li je $j<p$, treba postaviti *selm*='1', tako da se rezultat postavlja na *compn*. Poređenje $A[j]<min$ se dobija na izlazu komparatora *compa*. Ukoliko jeste, izvršavaju se naredbe: $min \leftarrow A[j]$ (*enm*='1' i *selr*='0'); $k \leftarrow j$ (*enk*='1' i *selk*='0'); Inkrementiranje *j* se vrši postavljanjem *enj*='1'. Upis u memoriju $A[k] \leftarrow A[i]$ se vrši postavljanjem *seld*='0' , *selr*='1', *selw*='1' i *we*='1';
- **S3**: Upis u memoriju $A[i] = min$ se vrši postavljanjem *seld*='1' , *selw*='0' i *we*. Za inkrementiranje registra *i* dovoljno je *eni*='1'.

6.2.7 Blok dijagram sistema

Na Sl. 15 su prikazani detalji sprege staze podataka i upravljačke jedinice.



Sl. 14



Sl. 15

6.2.8 VHDL opis

Organizacija VHDL projekta je usklađena sa sistemskim blok dijagramom sa Sl. 15. Staza podataka (***`Data_path`***) i upravljačka jedinica (***`Control_unit`***) su opisane na funkcionalnom nivou, a zatim objedinjene u IM blok. U vršnom modulu, ***`IM`*** (Sl. 11) kreirane su instance IM bloka i registarskog fajla, dok su RF brojač i multiplekser opisani na funkcionalnom nivou.

6.2.9 Simulacija

Radi lakšeg testiranja, u procesu za stimulanse napisana je procedura *rf_write* koja vrši upis inicijalnog sadržaja u registarski fajl:

```
-- procedura za upis inicijalnog sadrzaja u RF
procedure write_rf is
   -- A - niz koji treba upisati u RF
    type int_array is array(0 to 15) of integer;
variable A : int_array
                  := (22,71,19,2,115,48,201,22,50,111,72,9,98,82,228,189);
begin
     we <= '1';
     for i in 0 to 15 loop
       w_addr <= std_logic_vector(to_unsigned(i,6));
       w_data <= std_logic_vector(to_unsigned(A(i),8));
       wait for clk_period;
     end loop;
     we <= '0';
end ;
```

Procedura se nalazi u deklarativnom delu procesa i zbog toga su joj neposredno dostupni svi signali iz arhitekture i varijable iz istog procesa. To je razlog zašto ova procedura nema ni ulazne ni izlazne argumente. Sadržaj registarskog fajla je definisan nizom celih brojeva *A*, koji je deklarisan u samoj proceduri. U telu procedure, uz aktivan signal upisa u registarski fajl, *we*, redom se vrši upis elemenata niza *A*. Na port *w_addr* IM modula se postavlja adresa upisa, koja je jednaka indeksu elementa u nizu *A*, a na port *w_data* vrednost elementa (uz konverziju iz tipa *integer* u tip *std_logic_vector*). Vremenski razmak između dva uzastopna upisa je jednak trajanju periode taktnog signala, *clk_period*.

- ♦ ♦ ♦ -