# Mikroprocesorski sistemi

## Naziv projekta:

## *Watchdog* procesor

Vukašin Spasić 10354
Velimir Ilić 11056

# Sadržaj:

# *Watchdog* procesor

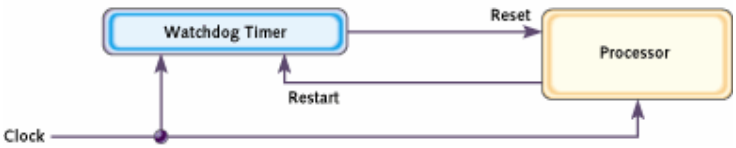
**Za one *embedded* sisteme koji ne mogu stalno biti nadgledani od strane ljudi, *watchdog* procesori mogu biti rešenje.**

Većina *embedded* sistema mora da bude nezavisno. Uglavnom nije moguće ili opravdano da se čeka neko ko će ih restartovati, ako se softver zaglavi. Neki *embedded* projekti, kao što su svemirske sonde, jednostavno nisu dostupne ljudskim operatorima. Ako njihov softver ikada zakaže, ti sistemi su zauvek onesposobljeni. U drugim slučajevima, brzina kojom operator može da resetuje sistem može da bude nedovoljna da zadovolji zahteve proizvoda.

*Watchdog* (pas čuvar) procesor je deo hardvera koji može da se koristi da automatski otkriva softverske anomalije, odnosno nepravilnosti i resetuje procesor ako se neka pojavi. Uopšteno govoreći, *watchdog* procesor je zasnovan na brojaču (counter) koji broji od nule do neke određene vrednosti ili obrnuto. Softver određuje vrednost brojača i periodično ga resetuje. Ako brojač dostigne tu vrednost (ili nulu) pre nego što ga softver resetuje, predpostavlja se da softver neispravno funkcioniše i formira se signal. Taj signal će resetovati procesor i softver koji se do tada izvršavao kao da ih je resetovao operator ili će izazvati prekid, a u okviru prekidne rutine program se upućuje na neki bezbedan deo. Često se ove dve aktivnosti kombinuju tako što se u okviru prekidne rutine :

- izvršavanje programa uputi na neki bezbedan deo programa
- testira se deo po deo sistema da bi se detektovala greška koja se memoriše
- zatim se resetuje ceo sistem

Na slici 1. se može videti uobičajen raspored. Kao što se vidi, *watchdog* je eksterni čip u odnosu na procesor. Ipak, može biti uključen u isti čip kao i procesor (CPU). Ovo se radi kod mnogih mikrokontrolera. U svakom slučaju, izlaz iz *watchdog* procesora je direktno povezan sa signalom reseta procesora.




Slika 1.

## Šutiranje psa

Proces resetovanja *watchdoga* se ponekad zove „šutiranje psa“ (kicking the dog). Odgovarajuća metafora je slučaj kada je čoveka napao opasan pas. Ako nastavi da šutira psa, on ne može da ga ujede. Ali on mora da nastavi da šutira psa u odgovarajućim intervalima da bi izbegao ujed. Slično, softver mora da resetuje *watchdog* procesor u odgovarajućem periodu, da ne bi bio resetovan.

## Softverske anomalije

*Watchdog* procesor može da izvuče sistem iz mnogo opasnih situacija. Ali, ako želimo da budemo efektivni, resetovanje *watchdog* procesora mora da se uzme u

obzir prilikom projektovanja softvera. Programeri moraju da znaju koje stvari mogu da krenu naopako sa njihovim softverom, i da omoguće da ih *watchdog* otkrije, ako se neka pojavi. Sistem može da se zaglavi iz mnogo razloga. Najjednostavnija je logička greška koja rezultuje izvršavanjem beskonačne petlje. Druga mogućnost je da neuobičajen broj *interrupt*-a stigne za vreme jednog prolaska kroz petlju. Bilo koje dodatno vreme provedeno u ISRu ( *Interrupt Service Routine*) je vreme za koje se ne izvršava glavna petlja. Kada se koriste *multitasking kerneli* može doći do zaglavljivanja. Na primer, grupa programa može da se zaglavi čekajući jedan drugog i neki spoljašnji signal koji je potreban jednom od njih, ostavljaju čitavu grupu programa da čeka do beskonačnosti. Ako su takve greške privremene, sistem može da radi savršeno tokom nekog vremenskog perioda nakon svakog reseta koje je izazvao *watchdog*. Ali, zato neispravan hardver može da dovede sistem do stalnog resetovanja. Iz tog razloga bilo bi mudro da se broji broj reseta koje je izazvao *watchdog* procesor, i da se prestane sa pokušajima posle nekog određenog broja reseta. Ako je *watchdog* ugrađen u mikrokontroler, možda neće biti dozvoljen (enabled) kada se sistem resetuje. Zato mora da se uključi za vreme inicijalizacije hardvera. Da bi se sprečilo da greška slučajno zabrani (disable) *watchdog*, hardver je obično tako projektovan da nije moguće isključiti *watchdog* kada se jednom dozvoli.

# Pregled specifikacije AMBA magistrale

Pošto watchdog procesor u ovom slučaju treba da bude prilagođen na AMBA magistralu, prvo da se bliže upoznamo sa njom. AMBA specifikacija definiše *on-chip* komunikacioni standard koji se odnosi na projektovanje visoko performansnih *embedded* mikrokontrolera. Amba specifikacijom se definišu tri različite magistrale:

- Advanced High-Performance Bus (AHB)
- Advanced System Bus (ASB)
- Advanced Peripheral Bus (APB)

### AHB magistrala

AHB je visoko performansna magistrala predviđena za rad na visokim frekvencijama koja se ponaša kao kičma magistrala. AHB podržava efikasnu procesorsku spregu, rad sa on-chip memorijama i off-chip eksternim memorijskim interfejsima koje se odlikuju malom potrošnjom.

### ASB magistrala

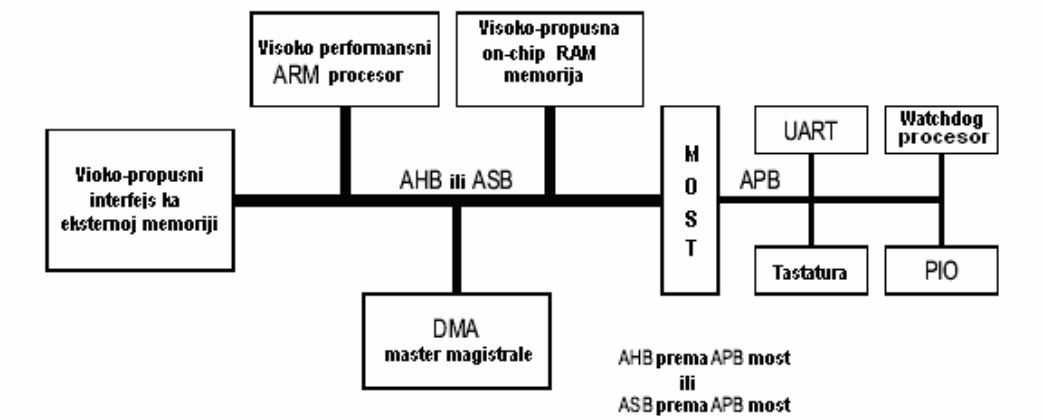
Visoko performansna magistrala predviđena za povezivanje sistemskih modula, koja se koristi kao alternativa AHB-u. ASB takođe podržava rad sa većim brojem procesora, on-chip memorijama, i off-chip eksternim memorijskim interfejsima koje se odlikuju malom potrošnjom.

### APB magistrala

APB je predviđena za spregu sa periferijama koje karakteriše mikro potrošnja. APB se može koristiti sa obe verzije sistemske magistrale AHB i ASB.

**Tipična struktura mikrokontrolera zasnovana na AMBA magistrali**

Jedna tipična struktura mikrokontrolera organizovanog oko visoko performansne sistemske AHB ili ASB magistrale i APB magistrale prikazan je na slici 2. Na sistemsku magistralu pored procesora (CPU-a) povezani su visoko-propusna *on-chip* RAM memorija, DMA kontroler magistrale, most za spregu sa APB i visoko-propusni memorijski interfejs za spregu sa spoljnom magistralom.




Slika 2. Tipični AMBA sistem sa obeleženim watchdog procesorom

| AMBA AHB | AMBA ASB | AMBA APB |
|---|---|---|
| Visoko performansna | Visoko performansna | mala potrošnja |
| Protočni rad | Protočni rad | lečovane adrese i upravljački signali |
| Magistrala sa većim brojem *master*-a | Magistrala sa većim brojem *master*-a | jednostavan interfejs |
| Paketni prenos | | pogodna za rad sa većim brojem periferija |
| Razbijene transakcije | | |

Pošto *watchdog* procesor predstavlja periferiju koja se nalazi na AMBA APB magistrali, sledi opširnija specifikacija AMBA APB magistrale.

# AMBA APB

APB je deo dvonivoske hijerajski organizovane AMBA magistrale. Ona se koristi za spregu sa periferijama čija je brzina rada relativno spora i u slučajevima kada se ne zahteva protočni princip rada. Kod zadnje verzije APB-a obezbeđeno je da se sve promene signala ostvaruju sa usponskom ivicom taktnog impulsa. Poboljšanja se odnose na sledeće:

- poboljšan je rad pri visokim frekvencijama
- poboljšanje je nezavisno od odnosa signal-pauza (faktora popune) taktne pobude
- statička vremenska analiza je pojednostavljena jer se koristi jedinstvena taktna pobuda
- ne postoje posebni zahtevi za automatsko ubacivanje testnih sekvenci
- bolja usklađenost kod primene *ASIC* kola iz razloga što se najveći broj njihovih internih registara taktuje na usponsku ivicu
- jednostavna integracija sa simulatorima koji rade ciklus po ciklus

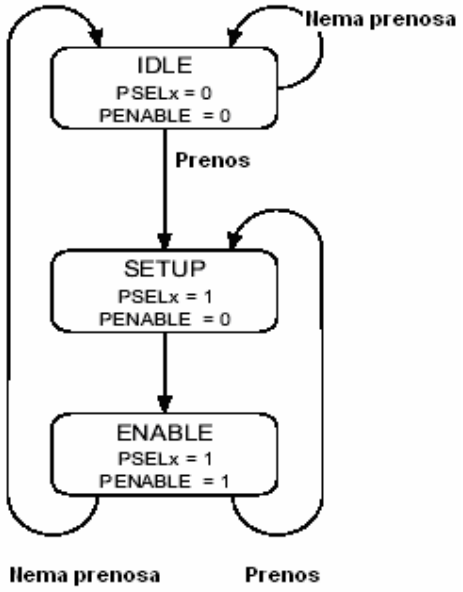
Ove promene u APB-u, takođe, uprošćavaju interfejs sa novim Advanced High-Performance Bus (AHB).

AMBA specifikacija je opisana:

- dijagramom stanja
- operacijom upisa
- operacijom čitanja

**Dijagram stanja**

Dijagram stanja prikazan na slici 3. se koristi za predstavljanje aktivnosti na periferijskoj magistrali.




Slika 3. Dijagram stanja

Kao što se vidi sa slike konačni automat prolazi kroz sledeća tri stanja:

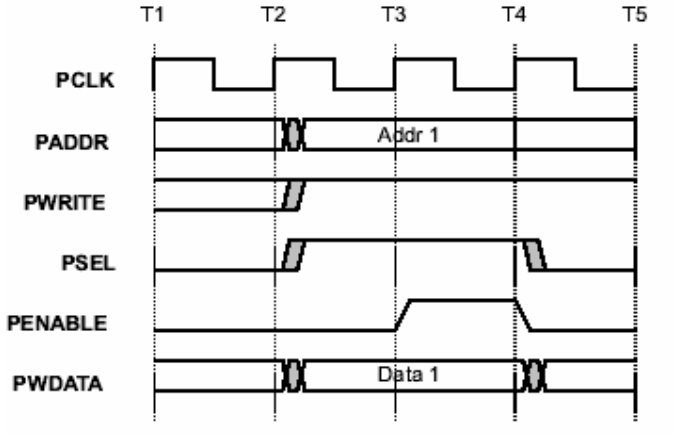
IDLE - Inicijalno stanje periferijske magistrale.

SETUP - Kada je potrebno da se ostvari prenos magistrala prelazi u SETUP stanje pri čemu se aktivira odgovarajući select signal PSELx. Magistrala ostaje u ovom stanju samo za jedan taktni interval i uvek prelazi u stanje ENABLE sa narednom usponskom ivicom taktnog impulsa.

ENABLE – U toku ovog stanja aktivira se signal PENABLE. Adresni i upravljački signali za upis i selekciju ostaju nepromenjeni u toku prelaza iz SETUP u ENABLE stanje. ENABLE stanje takođe traje jedan taktni interval i nakon ovog stanja magistrala se vraća u stanje IDLE za slučaj da se ne zahteva novi prenos. Alternativno, ako postoji zahtev za novim prenosom, magistrala prelazi u stanje SETUP. Karakteristično je da se na izlazima signala za adresu, upis i selekciju pojavljuju kratkotrajne smetnje (*glitch*) u trenutku prelaza iz stanja ENABLE u SETUP.

### Operacija upisa

Talasni dijagrami koji se odnose na operaciju upisa nalaze se na slici.

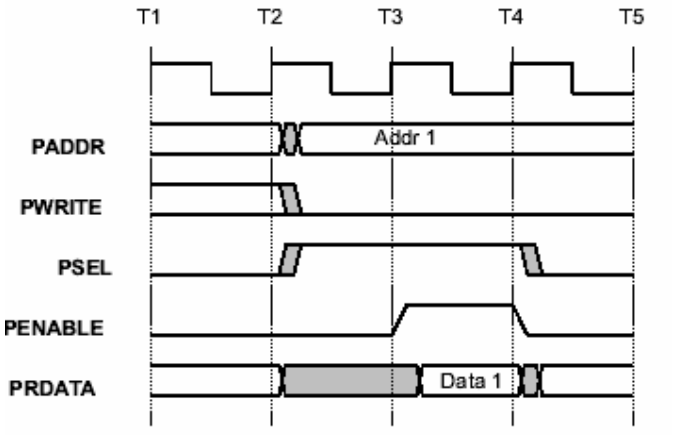



Slika 4. Operacija upisa

Operacija upisa počinje na taj način što se nakon usponske ivice taktnog impulsa postavljaju adrese, podaci koji se upisuju, i upravljački signali za upis i selekciju. Prvi taktni interval ove operacije se naziva SETUP ciklus. Sa sledećom ivicom taktnog impulsa aktivira se impuls PENABLE što ukazuje da se prešlo u stanje ENABLE. Signali za adrese, podaci i upravljane ostaju nepromenjeni u toku ENABLE ciklusa. Prenos se završava na kraju ovog ciklusa. Signal PENABLE se takođe deaktivira na kraju ovog prenosa. Selektorski signal takođe prelazi u stanje nisko što ukazuje da neposredno nakon toga može da sledi drugi prenos podataka ka istoj periferiji. Sa ciljem da se redukuje potrošnja signali za adrese i upis se ne menjaju nakon ovog prenosa sem ako se ne javi potreba za narednim pristupom. Protokol postavlja zahteve za korektan prelaz signala PENABLE.

### Operacija čitanja

Talasni dijagrami operacije READ prikazani su na slici.




Slika 5. Operacija čitanja

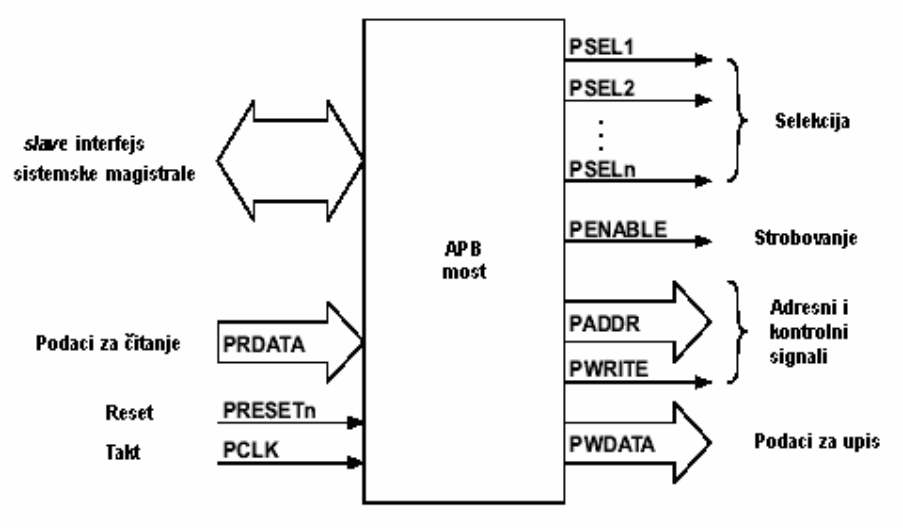
Talasni dijagrami operacije READ prikazani su na slici. Vremenski redosled generisanja signala za adrese, čitanje, selekciju i strobovanje je isti kao i kod operacije

upisa. U slučaju čitanja slave mora da generiše podatke u toku ciklusa ENABLE. Podaci se uzorkuju sa usponskom ivicom taktnog impulsa na kraju ciklusa.

## APB most

APB most je *master* APB magistrale, a *slave* za AHB magistralu. Na slici 6. je prikazan APB most interfejs

## Opis APB mosta




Slika 6. APB most

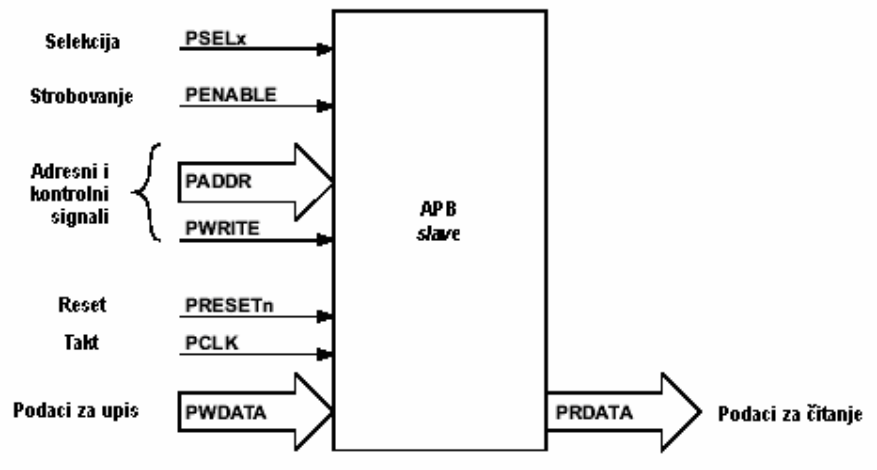
Ovaj most konvertuje prenose sa sistemske magistrale na prenose APB magistrale i obavlja sledeće funkcije:

- Lečuje adrese i zadržava ih u toku celokupnog prenosa
- Dekodira adrese i generiše periferijski selekcioni signal PSELx
- Samo jedan selekcioni signal je aktivan u toku jednog prenosa
- Aktivira linije za podatke na APB magistrali u toku operacije upisa
- Aktivira linije za podatke na sistemskoj magistrali kod operacije READ
- Generiše PENABLE signal

**APB *slave***

To su veoma jednostavni, ali fleksibilni interfejsi. Egzaktna implementacija interfejsa zavisi od stila projektovanja i velikog broja različitih opcija koje stoje na raspolaganju. Interfejs APB slave je prikazan na slici 7.

**Opis APB *slave*-a**




Slika 7. APB *slave*

Kod operacije upisa podaci se mogu lečovati u sledećim trenucima:

- U toku usponske ivice PCLK-a, kada je PSEL visoko
- U toku usponske ivice PENABLE kada je PSEL visoko

Signal selekcije PSEL, adresa PADDR, i signal za upis PWRITE se mogu kombinovati sa ciljem da se odredi sadržaj kog registra će biti ažuriran u toku operacije upisa.

Kod operacije upisa podaci se generišu na magistrali za podatke kada je PWRITE nisko i oba PSELx i PENABLE visoko. Vrednost na linijama na PADDR se koristi da odredi adresu registra koji se čita.

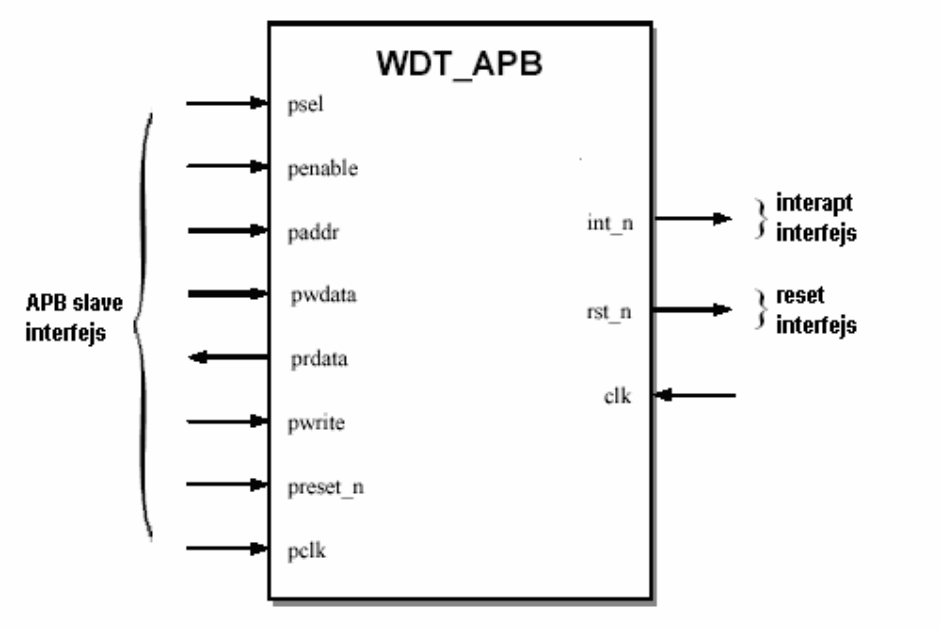
# Opis *watchdog* procesora

Jezgro *watchdog* procesora se sastoji od 16-bitnog brojača na dole (broji od veće ka manjoj vrednosti). Ima dva operaciona moda. Jezgro ima AMBA periferijski interfejs za povezivanje sa spoljnjim CPU-om.

**Osobine:**

- 16-bitni brojač na dole
- Programibilni *timeout* period
- Dva operaciona moda
- Mikroprocesorski interfejs kompatibilan sa AMBA APB interfejsom

**Simbolički prikaz:**




Slika 8. Simbolički prikaz

**Opis pinova:**

| Ime | Tip | Polaritet/Veličina bus-a | Opis |
|---|---|---|---|
| preset_n | I | Logička nula | **Reset:** Signal reseta APB magistrale |
| pclk | I | Usponski | **Sistemski takt:** APB takt se koristi da taktuje sve prenose |
| psel | I | Logička jedinica | **Selekcija:** Signal iz adresnog dekodera. Ovaj signal ukazuje da je *slave* uređaj selektovao APB most i da se zahteva prenos podataka |
| penable | I | Logička jedinica | **Strobovanje:** APB signal dozvole je u stanju jedinice tokom jednog ciklusa pclk-a da bi dozvolio prenos magistralom |
| pwrite | I | Logička nula | **Smer prenosa:** Signal smera APB prenosa. Kada je jedinica, označava upis, kada je nula označava čitanje |
| paddr | I | 16 | **Adresna magistrala:** APB adresna magistrala |
| pwdata | I | 16 | **Magistrala upisa podataka:** APB magistrala upisa podataka |
| prdata | O | 16 | **Magistrala čitanja podataka:** APB magistrala čitanja podataka |
| int_n | O | Logička nula | ***Interrupt*:** Ovaj signal je jedinica kada *watchdog* procesor odbroji do nule |
| rst_n | O | Logička nula | **Reset:** Ovaj signal je jedinica kada vrednost *watchdog* procesora dostigne nulu |
| clk | I | Usponski | **Takt procesora:** Svaka rastuća ivica ovog signala umanjuje vrednost brojača |

**Opis funkcionisanja:**

*Watchdog* procesor sadrži 16-bitni brojač i *wrapper* (spregu) sa APB interfejsom. *Watchdog* procesor mora da se isprogramira pre nego što može da se koristi. Programira se upisom početne vrednosti u *Load* registar i upisom kontrolne reči u *Config* registar. Mod se bira programiranjem *Config* registra.

**Operacioni modovi:**

Dostupna su dva moda funkcionisanja:

**Mod 1**

U ovom modu brojač se puni upisom početne vrednosti u *Load* registar. Procesor se aktivira automatski nakon upisa vrednost koja nije nula. Ako se upiše vrednost različita od nule dok brojač radi, onda se brojač trenutno puni novom vrednošću i počinje da odbrojava od nove vrednosti. Ako se upiše nulta vrednost, brojač i njegovi izlazi su isključeni. Kada brojač dostigne vrednost nula, biće generisan signal *interrupt*-a. Posle dostizanja nule procesor će ponovo upisati vrednost iz *Load* registra i nastaviti da odbrojava. Upisom u *Interrupt* registar je moguće isključiti *interrupt* signal. Ako se *interrupt* signal ne isključi, sledeće dostizanje nule će generisati reset signal. Brojač će se ponovo napuniti i restartovati odbrojavanje i isključiti signale na izlazima upisom u *Clear* registar.

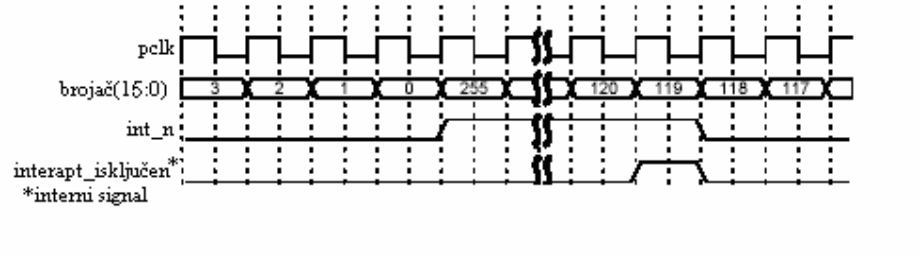
**Mod 2**

Ovaj mod je isti kao mod 1 sa jednom razlikom – *interrupt* je isključen. Posle prvog dostizanja vrednosti nula *watchdog* procesor će odmah generisati reset signal.

***Interrupt***

*Watchdog* može da se programira da generiše *interrupt* (a onda sistemski reset) kada odbroji do nule, odnosno kada istekne *timeout* period. Ako se ne očisti do vremena kada ponovo istekne *timeout* period, generiše sistemski reset. Ako se *watchdog* procesor restartuje u isto vreme kada *watchdog* dostigne nulu, *interrupt* se ne generiše.

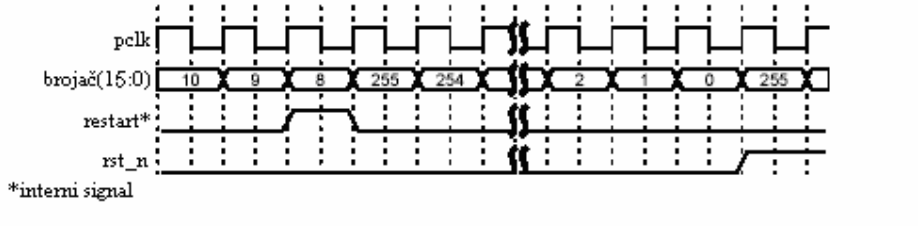
Slika prikazuje vremenski dijagram *interrupt*-a koji je generisan i onda isključen. *Interrupt* se isključuje upisom u *Interrupt* registar. *Interrupt* se takođe može isključiti upisom u *Clear* registar.




Slika 9. Generisanje i isključivanje *interrupt*-a

**Sistemski reset**

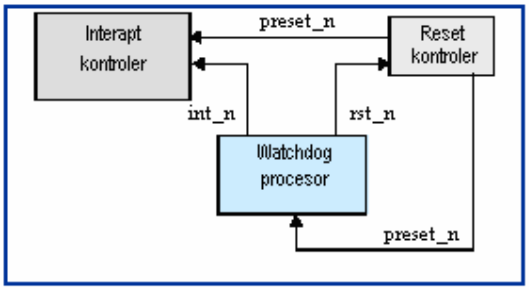
Kada je *watchdog* procesor u modu 1 posle generisanja *interrupt* signala, generiše se reset signal, ako brojač ponovo odbroji do nule, odnosno ako je *watchdog* procesor u modu 2 nakon prvog odbrojavanja brojača do nule generiše se reset signal. Ako se *watchdog* procesor restartuje u isto vreme kada brojač stigne do nule, sistemski reset se ne generiše. Reset se takođe može isključiti upisom u *Clear* registar.




Slika 10. Restartovanje brojača i generisanje reseta

***Interrupt* i reset kontroleri:**

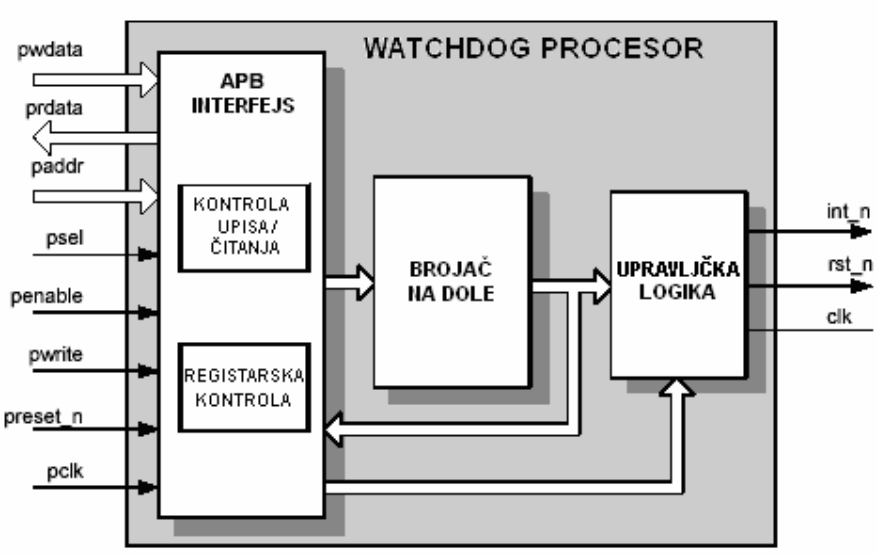
Generisani *interrupt* signal se vodi na *interrupt* kontroler. Generisani reset signal se vodi na reset kontroler, koji zauzvrat generiše reset za komponente u sistemu. *Watchdog* može da se resetuje nezavisno od drugih komponenti.




Slika 11. Veza *watchdog* procesora sa *interrupt* i reset kontrolerom

## Blokovi watchdog procesora

### Blok dijagram:




Slika 12. Blok dijagram

## APB interfejs

APB interfejs omogućava upravljanje i programiranje *watchdog* procesora sprežući ga sa APB magistralom. Adresni PADDR, signali za čitanje PRDATA i upis PWDATA su 16-bitni. U adresnom signalu četiri bita najmanje težine adresiraju *watchdog* procesor dok ostali bitovi adresiraju registre u samom *watchdog* procesoru. Takođe, APB interfejs vrši registarsku kontrolu, odnosno upis i čitanje registara, koji se nalaze u ovom bloku. Povezan je sa brojačem i kontrolnom logikom.

### Opis registara:

***Load* registar:**

*Load* registar sadrži početnu vrednost procesora. Registar se koristi kao vrednost za ponovni upis. Ovo je registar za upis i čitanje.

***Interrupt* registar:**

Upisivanje u *Interrupt* registar isključuje *interrupt* koji je generisao procesor i ponovo puni procesor i restartuje odbrojavanje. Ovo je registar za upis.

***Clear* registar:**

Upisivanje u *Clear* register ponovo puni procesor i restartuje odbrojavanje i isključuje *interrupt* i reset signal koji je generisao procesor. Ovo je registar upisa.

***Config* registar:**

*Config* registar daje mod operacije *watchdog*-a. Ovo je registar za upis i čitanje.

Izbor modova:
-mod 1 - nulta vrednost
-mod 2 – jedinična vrednost.

***Status* registar**

U Statusni registar se upisuje trenutno stanje u kome se nalazi brojač. Čitanjem iz Statusnog registra softver može da odredi dokle je odbrojao brojač i po potrebi odrediti novu početnu vrednost za *watchdog* procesor.

## Brojač

*Watchdog* procesor broji od početne (*timeout*) vrednosti naniže do nule. Naravno, da bi brojač bio u mogućnosti da broji, na *clk* ulazu watchdog procesora mora da postoji taktna pobuda. Taktna pobuda može biti eksterna ili se može koristiti takt APB magistrale. Kada brojač dostigne nulu, u zavisnosti od izabranog moda, generišu se ili *interrupt* ili reset signal. Takođe, u tom trenutku, brojač se ponovo puni izabranom *timeout* vrednošću i brojač nastavlja da odbrojava. Korisnik može da restartuje brojač na početnu vrednost. To se obavlja upisom u *Load* registar u bilo kom trenutku.

**Vrednosti *timeout* perioda**

*Watchdog* procesor može da se proizvoljno programira po pitanju početne vrednosti odnosno *timeout* perioda. Vrednosti koje mogu da se izaberu su ograničene širinom brojača koja iznosi 16 bita što znači da je najveća moguća vrednost koja se može postaviti 65535. Vrednost je ograničena u vreme konfigurisanja, i vrednosti veće od širine brojača se ne prihvataju.

## Upravljačka logika

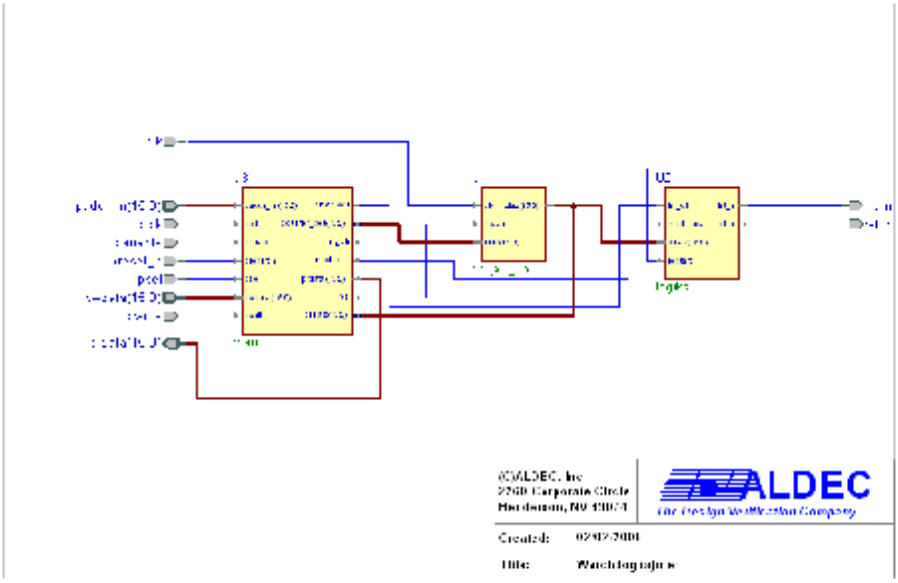
U ovom bloku vrši se generisanje izlaznih signala *watchdog* procesora u zavisnosti od stanja u kome se nalazi brojač i moda u kome *watchdog* procesor operiše. Ako je *watchdog* procesor postavljen u mod 1 prilikom prvog dostizanja vrednosti nula brojača generiše se *interrupt* signal, a nakon drugog dostizanja vrednosti nula signal reseta. U modu 2 se nakon prvog dostizanja vrednosti nula brojača generiše signal reseta. Takođe, upisom u *Clear* i *Interrupt* registar mogu se isključiti generisani *interrupt* i reset signali.

# Predlog realizacije watchdog procesora u VHDL-u

*Watchdog* procesor je opisan u programskom jeziku VHDL jedan od najkorišćenijih softverskih aplikacija za hardversko projektovanje. Za razvoj i testiranje koda korišćen je programski paket Activ-HDL.

U strukturi postoje tri entiteta:

- entitet „mem“
- entitet „brojac_16“
- entitet „logika“




Slika 13. Kompletna šema *watchdog* procesora

### 1. Entitet „mem“

„Mem“ sadrži interfejs za komunikaciju sa APB magistralom i omogućava upis i čitanje iz registarske memorije koju takođe sadrži. U memoriji se nalazi pet registara:

- *Load* registar (upisuje se početno stanje brojača)
- *Interrupt* registar (upisom u njega se prekida *interrupt* signal koji je generisao procesor)
- *Clear* registar (upisom u njega se brojač ponovo puni početnom vrednošću i restartuje brojanje)
- *Config* registar (definiše mod rada *watchdog* procesora, bez ili sa *interrupt* signalom)
- *Status* registar (sadrži trenutno stanje brojača)


Slika 14. Entitet „mem“

Listing koda:

```
1    library IEEE;
2    use IEEE.STD_LOGIC_1164.all;
3
4    entity mem is
5        port(
6            pclk : in STD_LOGIC;
7            psel : in STD_LOGIC;
8            penable : in STD_LOGIC;
9            pwrite : in STD_LOGIC;
10           preset_n : in STD_LOGIC;
11           paddr_in : in STD_LOGIC_VECTOR(15 downto 0);
12           pwdata : in STD_LOGIC_VECTOR(15 downto 0);
13           brulaz : in STD_LOGIC_VECTOR(15 downto 0);
14           clear_sel : out STD_LOGIC;
15           int_clr : out STD_LOGIC;
16           mod_sel : out STD_LOGIC;
17           res : out STD_LOGIC;
18           counter_sel : out STD_LOGIC_VECTOR(15 downto 0);
19           prdata : out STD_LOGIC_VECTOR(15 downto 0)
20           );
21   end mem;
22
23   --// End of automatically maintained section
24
25   architecture mem of mem is
26   begin
27       p1:process(pclk,psel,penable,pwrite,
28       preset_n,paddr_in,pwdata,brulaz) is
29       begin
30       if (pclk'event and pclk='1') then
31           if preset_n='1' then
32               counter_sel<="0000000000000000";
33               int_clr<'0';
34               clear_sel<'0';
35               mod_sel<='0';
36           else
37               if (paddr_in(3)='1' and paddr_in(2)='0' and
38                   paddr_in(1)='1' and paddr_in(0)='0' and psel='1') then
39                   if pwrite '1' then
40                       if penable='1' then
41                           if paddr_in "0000000001001010" then
42                               mod_sel<pwdata(0);
43                               res<='0';
44                           else
45                               null;
46                           end if;
47                           if paddr_in "0000000000011010" then
48                               counter_sel<=pwdata;
49                               res<='1';
50                           else
51                               null;
52                           end if;
53                           if paddr_in="0000000000101010" then
54                               int_clr<=pwdata(0);
55                               res<='1';
56                           else
57                               null;
58                           end if;
59                           if paddr_in="0000000000111010" then
60                               clear_sel<=pwdata(0);
61                               res<'1';
```

```
62                          else
63                              null;
64                          end if;
65                      else
66                          if brulaz "0000000000000000" then
67                              rea<='1';
68                          else
69                              rea<='0';
70                          end if;
71                          int_clr<='0';
72                          clear_sel<='0';
73                      end if;
74                  else
75                      int_clr<='0';
76                      clear_sel<='0';
77                  end if;
78              else
79                  if brulaz="0000000000000000" then
80                      rea<='1';
81                  else
82                      rea<='0';
83                  end if;
84                  int_clr<='0';
85                  clear_sel<='0';
86
87              end if;
88          end if;
89  end if;
90      end process p1;
91
92          p2 : process (penable, pwrite, pwrite, paddr_in) is
93          begin
94          if penable'event and penable '1' then
95              if psel='1' and pwrite='0' and paddr_in="0000000000110000"
    then
96                  prdata <= brulaz;
97              end if;
98          end if;
99           if  penable='0' then
100            prdata <="XXXXXXXXXXXXXXXX";
101         end if;
102
103     end process;
104
105 end architecture mem;
106
```

**2. Entitet „brojac_16“**

Ovaj entitet predstavlja 16-bitni brojač koji na ulazu prima stanje iz *Load* registra od koga počinje brojanje. Startuje brojanje kada signal restart promeni vrednost iz logičke jedinice u nulu. Na izlazu daje trenutno stanje brojača.

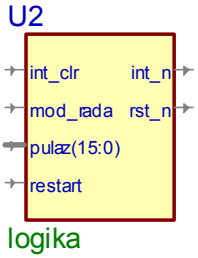

Slika 15. Entitet „brojac_16“

Listing koda:

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.all;
4   use IEEE.STD_LOGIC_UNSIGNED.all;
5
6   entity brojac_16 is
7    port(
8         clk : in STD_LOGIC;
9         restart : in STD_LOGIC;
10        ulaz : in STD_LOGIC_VECTOR(15 downto 0);
11        izlaz : out STD_LOGIC_VECTOR(15 DOWNTO 0)
12        );
13  end brojac_16;
14  architecture brojac_16 of brojac_16 is
15  signal broj:std_logic_vector(15 downto 0);
16  begin
17  p2:process(clk,restart) is
18  begin
19  if restart='1' then
20      broj<= ulaz;
21        izlaz<=ulaz;
22  elsif clk'event and clk='0' then
23      broj<=broj-1;
24      izlaz<=broj-1;
25  else
26      null;
27  end if;
28  end process p2;
29  end brojac_16;
30
```

### 3. Entitet „logika“

Ovaj entitet kontroliše izlaze *watchdog* procesora i to u zavisnosti od stanja brojača i vrednosti upisanih u *Interrupt*, *Clear* i *Config* registar. Ako je na izlazu brojača vrednost nula logika generiše *interrupt* signal koji ostaje aktivan dok se brojač resetuje i ponovo odbrojava od početne vrednosti i, u slučaju da ponovo odbroji do nule generiše se i reset signal, ako je *watchdog* procesor u modu 1. U modu 2 nakon prvog dostizanja vrednosti nula brojača generiše se reset signal, dok je *interrupt* isključen.




Slika 16. Entitet „logika“

Listing koda:

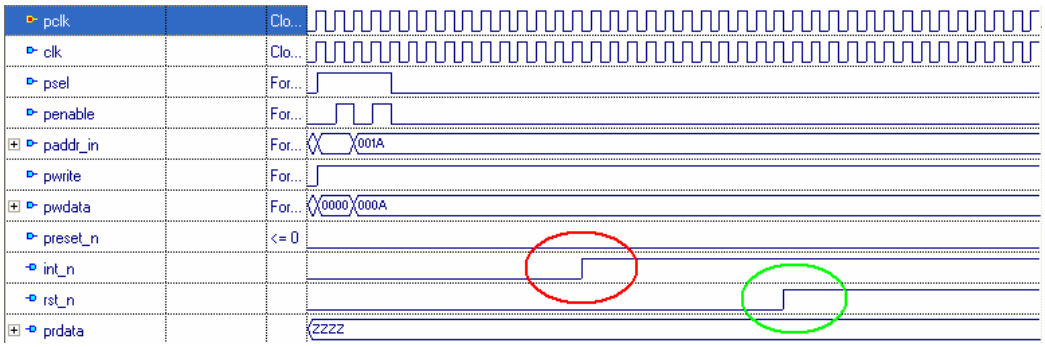
```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.all;
3   use IEEE.STD_LOGIC_unsigned.all;
4
5   entity logika is
6     port(
7          int_clr : in STD_LOGIC;
8          mod_rada : in STD_LOGIC;
9          restart : in STD_LOGIC;
10         pulaz : in STD_LOGIC_VECTOR(15 downto 0);
11         int_n : out STD_LOGIC:='0';
12         rst_n : out STD_LOGIC:='0'
13         );
14  end logika;
15
16  architecture logika of logika is
17  begin
18
19  p1:process (int_clr,restart,mod_rada, pulaz) is
20  variable pass:std_logic_vector(1 downto 0): "00";
21  begin
22
23       l1: if restart='1' or int_clr='1' then
24            int_n<='0';
25            rst_n<='0';
26            pass:="00";
27       elsif pulaz="0000000000000000" then
28            l3: if pass="00" and mod_rada='0' then
29                 int_n<='1';
30
31                elsif pass="01" and mod_rada='0' then
32                rst_n<='1';
33                elsif mod_rada='1' then
34                rst_n<='1';
35                int_n<='0';
36                  end if l3;
37            pass:=pass+1;
38          end if l1;
39  end process p1;
40  end logika;
41
```

**Listing koda *watchdog* procesora:**

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3
4
5   entity watchdog1 is
6     port(
7           clk : in STD_LOGIC;
8           pclk : in STD_LOGIC;
9           penable : in STD_LOGIC;
10          preset_n : in STD_LOGIC;
11          psel : in STD_LOGIC;
12          pwrite : in STD_LOGIC;
13          paddr_in : in STD_LOGIC_VECTOR(15 downto 0);
14          pwdata : in STD_LOGIC_VECTOR(15 downto 0);
15          int_n : out STD_LOGIC;
16          rst_n : out STD_LOGIC;
17          prdata : out STD_LOGIC_VECTOR(15 downto 0)
18    );
19  end watchdog1;
20
21  architecture watchdog1 of watchdog1 is
22
23  component brojac16
24    port (
25          clk : in STD_LOGIC;
26          restart : in STD_LOGIC;
27          ulaz : in STD_LOGIC_VECTOR(15 downto 0);
28          izlaz : out STD_LOGIC_VECTOR(15 downto 0)
29    );
30  end component;
31  component logika
32    port (
33          int_clr : in STD_LOGIC;
34          mod_rada : in STD_LOGIC;
35          pulaz : in STD_LOGIC_VECTOR(15 downto 0);
36          restart : in STD_LOGIC;
37          int_n : out STD_LOGIC := '0';
38          rst_n : out STD_LOGIC := '0'
39    );
40  end component;
41  component mem
42    port (
43          brulaz : in STD_LOGIC_VECTOR(15 downto 0);
44          paddr_in : in STD_LOGIC_VECTOR(15 downto 0);
45          pclk : in STD_LOGIC;
46          penable : in STD_LOGIC;
47          preset_n : in STD_LOGIC;
48          psel : in STD_LOGIC;
49          pwdata : in STD_LOGIC_VECTOR(15 downto 0);
50          pwrite : in STD_LOGIC;
51          clear_set : out STD_LOGIC;
52          counter_set : out STD_LOGIC_VECTOR(15 downto 0);
53          int_clr : out STD_LOGIC;
54          mod_set : out STD_LOGIC;
55          prdata : out STD_LOGIC_VECTOR(15 downto 0);
56          res : out STD_LOGIC
57    );
58  end component;
59
60  signal NET30 : STD_LOGIC;
61  signal NET46 : STD_LOGIC;
```

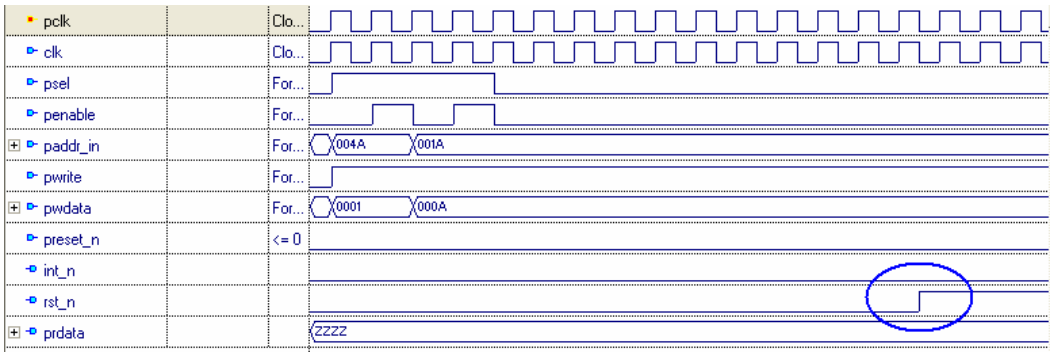
```
62    signal NET54 : STD_LOGIC;
63    signal NET62 : STD_LOGIC;
64    signal BUS100 : STD_LOGIC_VECTOR (15 downto 0);
65    signal BUS38 : STD_LOGIC_VECTOR (15 downto 0);
66
67    begin
68
69    U1 : brojac 16
70      port map(
71            clk => clk,
72            izlaz -> BUS100,
73            restart -> NET62,
74            ulaz  > BUS38
75      );
76
77    U2 : logika
78      port map(
79            int_clr -> NET46,
80            int_n -> int_n,
81            mod_rada  > NET54,
82            pulaz  > BUS100,
83            restart -> NET30,
84            rst_n -> rst_n
85      );
86
87    U3 : mem
88      port map(
89            brulaz  > BUS100,
90            clear set  > NET30,
91            counter set -> BUS38,
92            int clr -> NET46,
93            mod_sel => NET54,
94            paddr_in -> paddr_in,
95            pclk -> pclk,
96            penable  > penable,
97            prdata  > prdata,
98            preset n -> preset n,
99            psel -> psel,
100           pwdata => pwdata,
101           pwrite -> pwrite,
102           res -> NET62
103     );
104
105
106   end watchdog1;
107
```

**Izgled simulacije rada *watchdog* procesora u modu 1:**



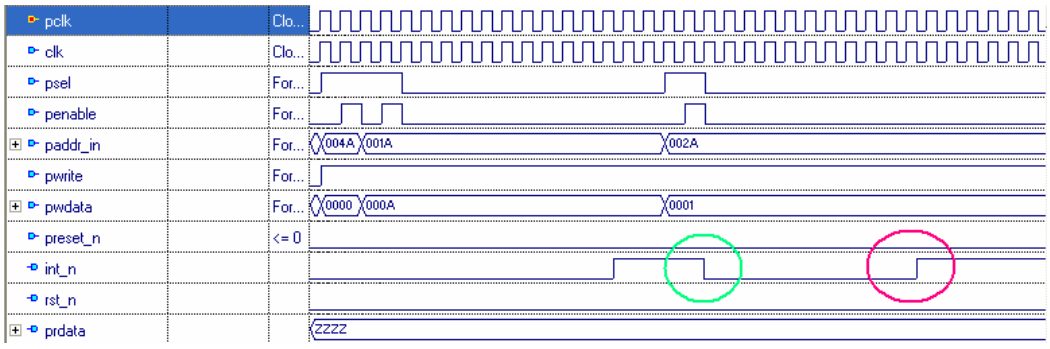

Slika 17.

**Izgled simulacije rada *watchdog* procesora u modu 2:**



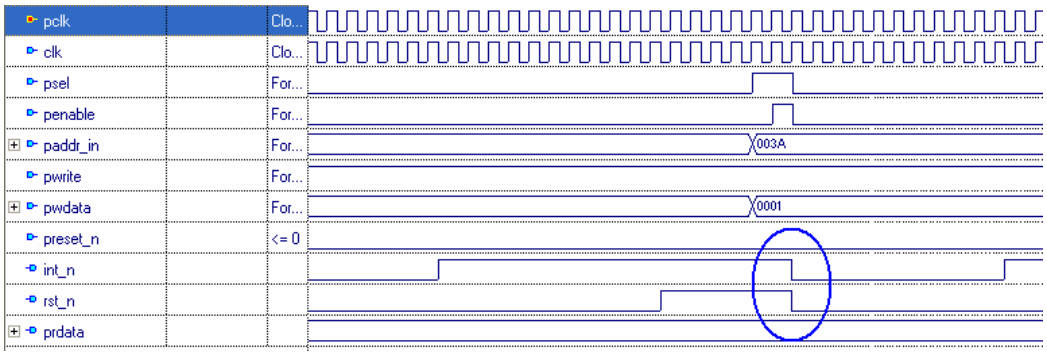

Slika 18**.**

**Izgled simulacije u slučaju isključivanja *interrupt* signala:**



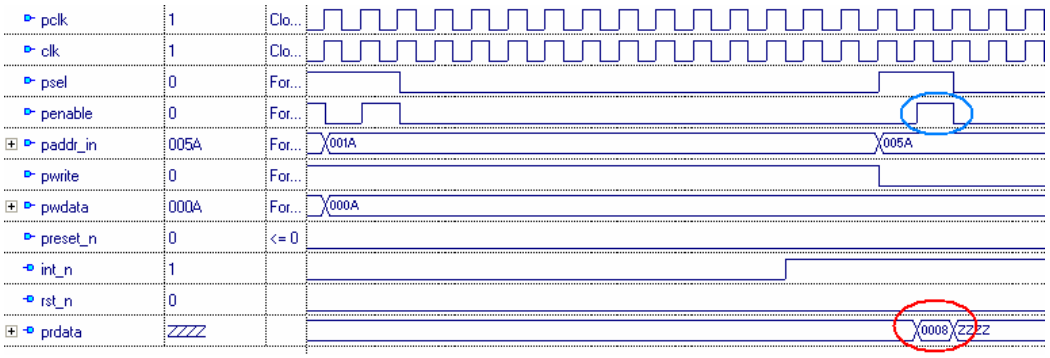

Slika 19.

**Izgled simulacije u slučaju čišćenja *interrupt* i reset signala:**




Slika 20.

**Izgled simulacije u slučaju čitanja trenutnog stanja brojača:**



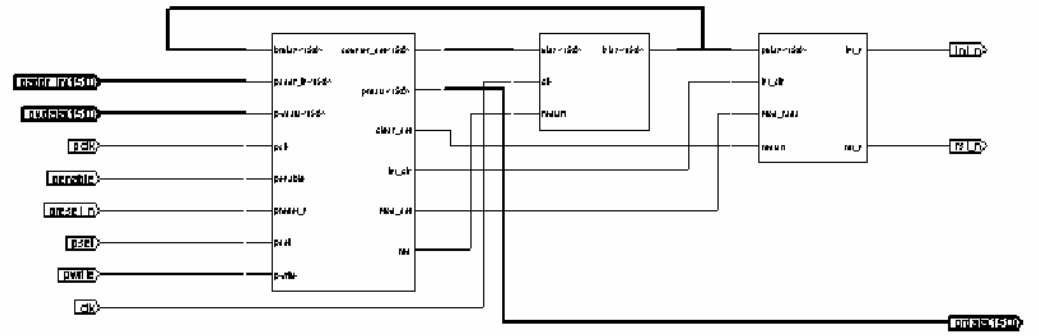

Slika 21.

**SINTEZA I IMPLEMENTACIJA KOLA**

Kolo je sintetizovano u programskom paketu firme *XILINX* ISE6.3i. Ovo je programski paket firme *XILINX* za razvoj aplikacija baziranih na njihovim CPLD i FPGA kolima.

Za implementaciju *watchdog* procesora iskorišćeno je FPGA kolo iz *XILINX*-ove familije SPARTAN2 koje nosi oznaku XC2S30-VQ15.

Na slici 21. prikazana je šema sinteze kola na najvišem nivou, koju je izgenerisao program *Xilinx ECS*. Na šemi se mogu uočiti osnovni blokovi od kojih se kolo sastoji, gde se naravno u programu može analizirati svaki blok dubinski do nivoa samih gejtova.



Slika 21.

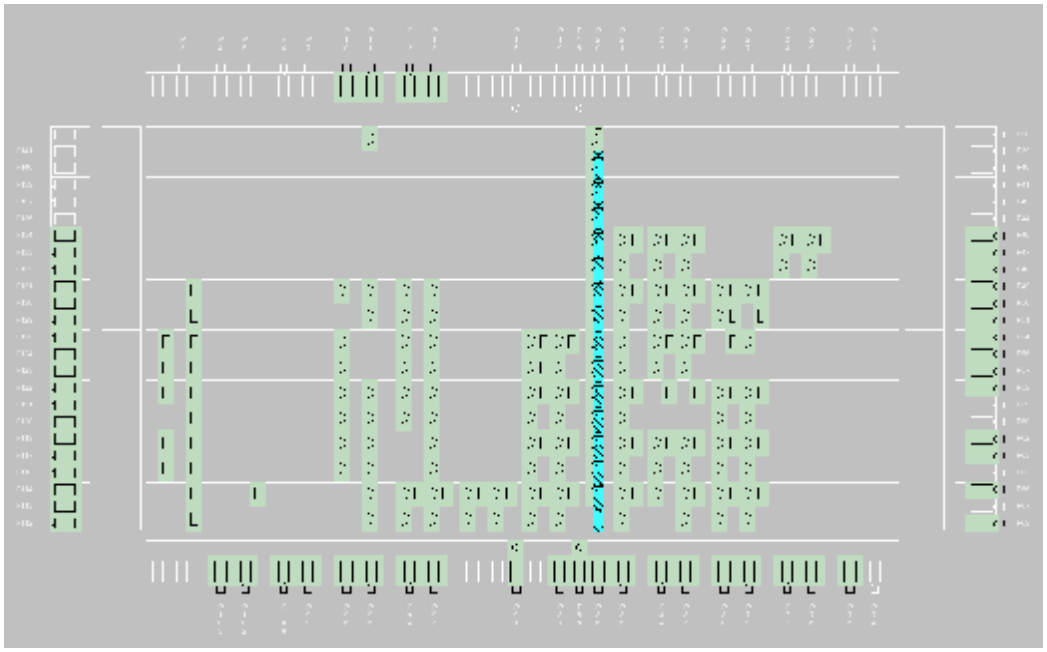
Nakon toga prešlo se na samu implementaciju kola na konkretnom FPGA kolu. Sledeći listing nam pokazuje dobijene rezultate:

```
Design Information
------------------
Command Line   : E:/Xilinx/bin/nt/map.exe -intstyle ise -p xc2s15-tq144-6 -cm
area -pr b -k 4 -c 100 -tx off -o watchdog1_map.ncd watchdog1.ngd watchdog1.pcf
Target Device  : x2s15
Target Package : tq144
Target Speed   : -6
Mapper Version : spartan2 -- $Revision: 1.16.8.2 $
Mapped Date    : Tue Apr 04 14:35:51 2006

Design Summary
--------------
Number of errors:      0
Number of warnings:   35
Logic Utilization:
  Total Number Slice Registers:      58 out of    384   15%
    Number used as Flip Flops:                38
```

```
    Number used as Latches:                20
  Number of 4 input LUTs:          120 out of    384   31%
Logic Distribution:
    Number of occupied Slices:                75 out of    192   39%
    Number of Slices containing only related logic:     75 out of     75  100%
    Number of Slices containing unrelated logic:        0 out of     75    0%
       *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:         121 out of    384   31%
      Number used as logic:              120
      Number used as a route-thru:         1
   Number of bonded IOBs:          54 out of     86   62%
      IOB Flip Flops:                  16
      IOB Latches:                     16
   Number of GCLKs:                2 out of      4   50%
   Number of GCLKIOBs:             2 out of      4   50%

Total equivalent gate count for design:  1,473
Additional JTAG gate count for IOBs:  2,688
```

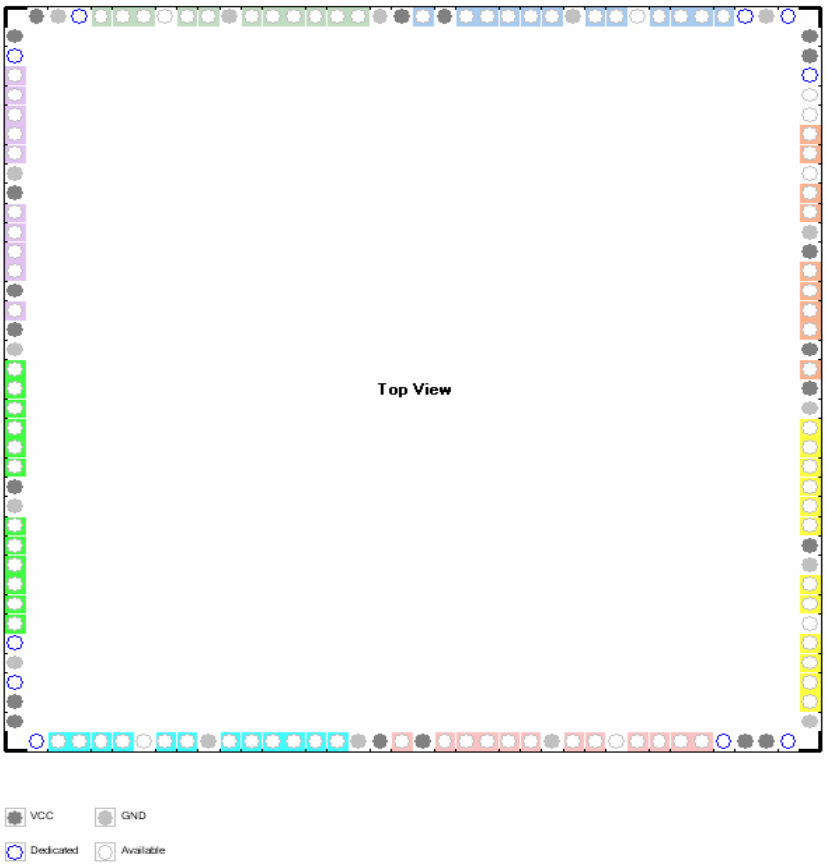
Na osnovu dobijenih podataka za nas je bitno da se na kraju vidi da je implemetnirano kolo zauzelo 1473 ekvivalentnih gejtova, kao i da je pridodato još 2688 gejta koja su neophodna za realizaciju JTAG logike kojom se vrši testiranje i programiranje FPGA kola. Kao još jedan važan podatak treba istaći da je maksimalna radna frekvencija kola 147.059 MHz.

Na slici se može videti unutrašnja struktura dobijena nakon implementacije kola gde se vide zauzeti blokovi unutar kola. Slika je generisana u *Xilinx Floorplanner*-u editoru koji je okviru paketa ISE zadužen za sam fizički razmeštaj ćelija. Slika prikazuje sam razmeštaj ćelija gde se može videti i koji pinovi kola su iskorišćeni. Iako se zapaža da izvodi kola nisu grupisani, to je da bi se postiglo približno isto kašnjenje veza do svih pinova.



Slika 22.

Na slici 23. prikazan je i sam fizički raspored pinova na FPGA kolu.




Slika 23.

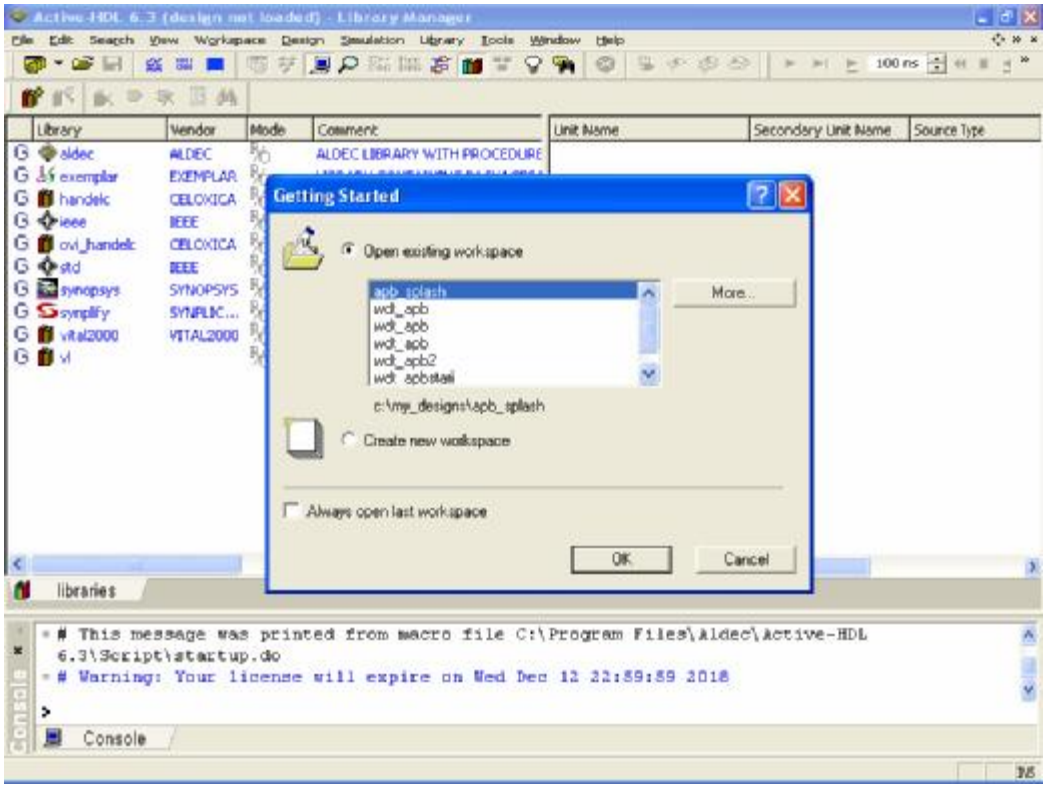
**Laboratorijska vežba**

Proveriti ispravnost rada *watchdog* procesora u oba moda:

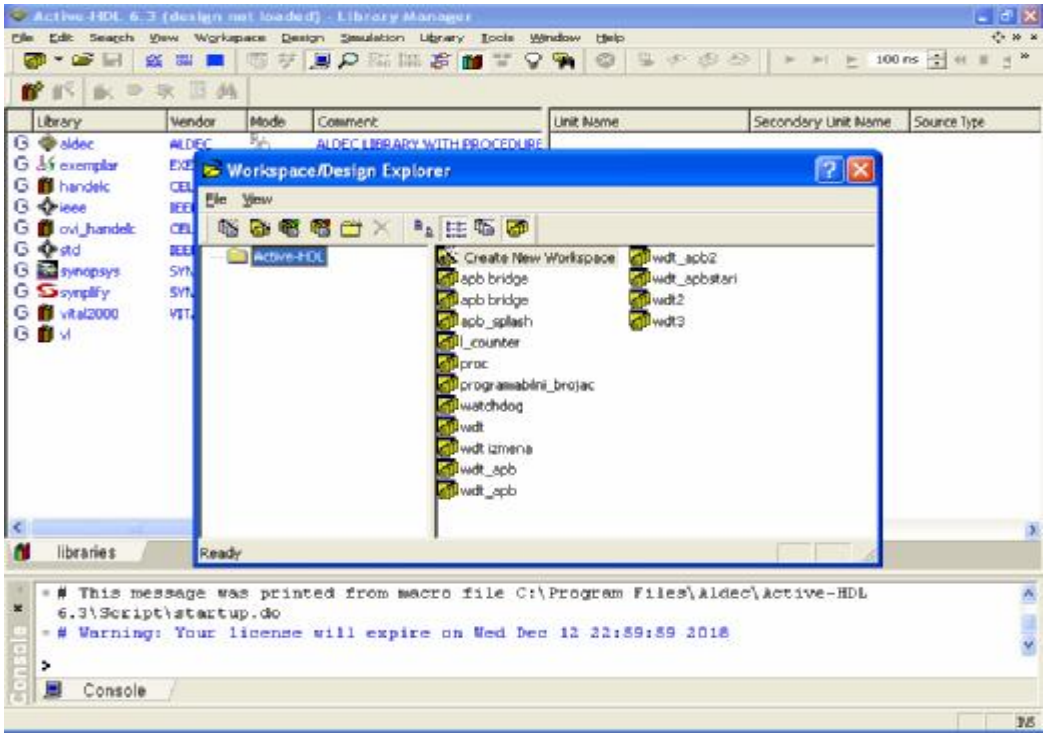
Dvoklikom na ikonu Activ-HDL-a (Slika 1.) pokrenuti program:


Slika 1.

Klikom mišem na polje *More...* (Slika 2.) izabrati dizajn APB_SPLASH (Slika 3.)



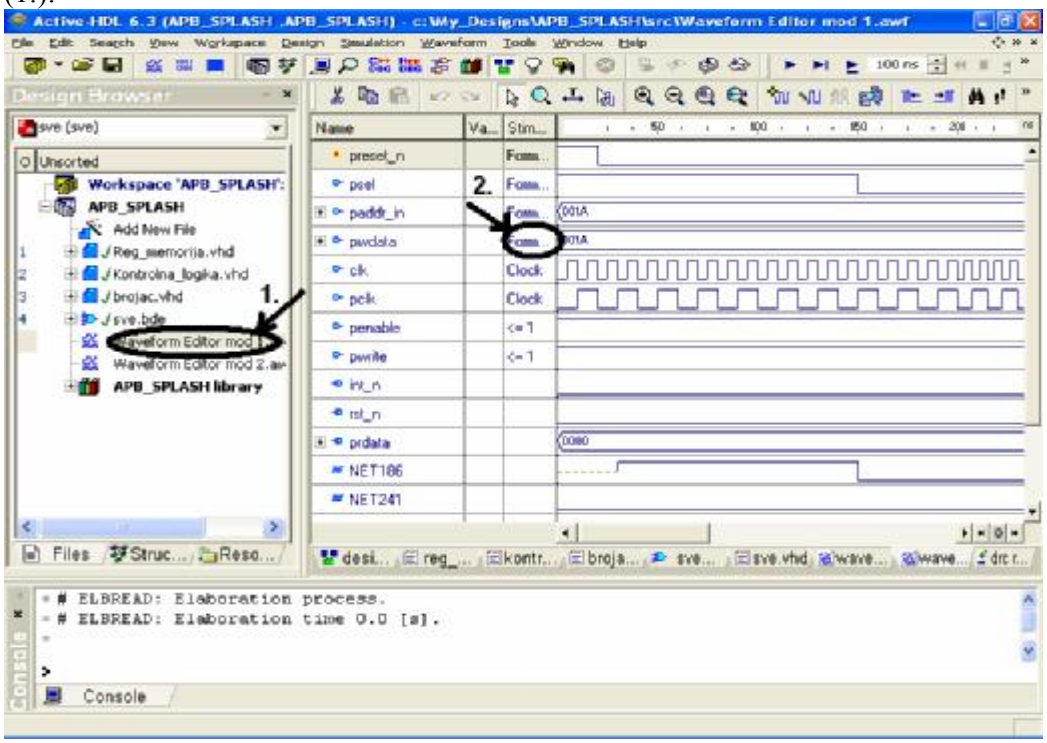

Slika 2.




Slika 3.

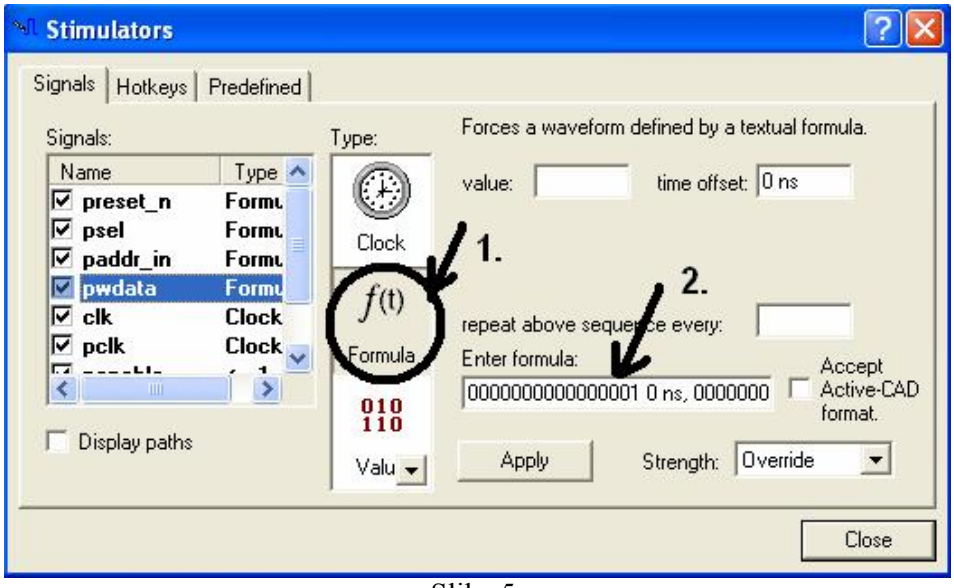
**1. MOD 1**

U okviru novo otvorenog radnog prostora (Slika 4.)izabrati *Waveform editor mod 1* (1.):




Slika 4.

Proveriti da li su na listi signali: preset_n, psel, paddr_in, pwdata, clk, pclk, penable, pwrite, int_n, rst_n, prdata. Da bi *watchdog* radio u modu 1 potrebno je prvo upisati nultu vrednost u *Config* registar, a zatim upisati početnu (*timeout*) vrednost u *Load* registar. Dvoklikom na polje signala pwdata u koloni *Stimulator* (2.) otvoriti prozor *Stimulators*.




Slika 5.

Klikom na tip *Formula* (1.) uneti formulu u polje *Enter Formula* (2.) uneti formulu oblika: 0000000000000000 (16 nula) 0 ns, "početna vrednost" 50 ns. Pokrenuti simulaciju. Da li je generisan *interrupt* i u kom trenutku. Da li je generisan reset i u kom trenutku. Ako nije zašto nije?

**2. MOD 2**

Sada u okviru radnog prostora izabrati *Waveform editor mod 2*:

Ponoviti postupak i sada u polje *Enter Formula* uneti sledeću formulu: 0000000000000001 (15 nula i jedinica) 0 ns, "početna vrednost" 50 ns. Pokrenuti simulaciju. Da li je generisan *interrupt* i u kom trenutku. Da li je generisan reset i u kom trenutku. U čemu je razlika između moda 1 i moda 2?

**Student 1:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 2:**

Mod 1:

Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 3:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 4:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 5:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 6:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____

Trenutak generisanja reseta_____

**Student 7:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 8:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 9:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 10:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 11:**

Mod 1:
Početna vrednost_____

Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 12:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 13:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 14:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 15:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Student 16:**

Mod 1:
Početna vrednost_____
Trenutak generisanja *interrupt*a_____
Trenutak generisanja reseta_____

Mod 2:
Početna vrednost_____
Trenutak generisanja reseta_____

**Literatura:**


1. **Introduction to Watchdog Timers by Michael Berr**
2. **AMBA Specification (Rev 2.0) by ARM Limited**
3. **Watchdog Timer Datasheet by Evatronix SA**