PROJEKTIRANJE UGRADBENIH RAČUNALNIH SUSTAVA

Materijali za predavanja

MODELIRANJE DIGITALNIH SUSTAVA

Zašto je potrebno modelirati digitalni sustav?

Krajnji cilj razvojnog rada

zaraditi novac <=> biti konkurentan na tržištu

Kako postići taj cilj?

- napraviti sustav koji dobro služi nekoj svrsi => *functionality*
- razviti sustav u što kraćem vremenu => *time to market*
- postići najveću moguću pouzdanost => *reliability*
- napraviti što manje grešaka => *debuging*
- po mogućnosti, prodati pojedine dijelove sustava više puta => *reuse*

Vrijeme razvoja posljednjih godina ima sve veću težinu; ***short time to market***

Modeliranje digitalnih sustava pomoću jezika za opis sklopovlja omogućuje

- "brušenje" tehničkog zahtjeva
- provjeru (simulacijom) da li sustav opisan tehničkim zahtjevom zadovoljava (u interakciji s okolinom)
- djelomično automatiziranje pojedinih faza dizajna
- verificiranje dizajna (npr. provjere kašnjenja razvijenog sklopa)
- jednostavniju izradu dokumentacije
- korištenje pojedinih podsustava u drugim projektima

- modeliranje je formalno
- sustav se modelira hijerarhijski

  => smanjuje se broj detalja u pojedinim dijelovima razvoja
  (slično kao u jeziku C)

JEZICI ZA OPIS DIGITALNIH SUSTAVA

HDL → Hardware Description Language(s)

- služe za formalno modeliranje digitalnih sustava

Popularni jezici: VHDL, Verilog, System C, Abel

VHDL → VHSIC HDL
↓
Very High Speed Integrated Circuits

Povijesni razvoj VHDL jezika

1981. - začetak VHDL-a u sklopu VHSIC programa USA-DoD
- nastalo zbog potrebe da standardizira opis sklopova, dokumentacija i verifikacija

1983. - počeo projekt izrade prve verzija VHDL jezika - IBM, TI i Intermetrics

1985. - gotova prva verzija VHDL jezika

1986. - IEEE (*The Institute of Electrical and Electronic Engineers*) počeo izradu standarda

1987. - IEEE Standard 1076-1987 => **VHDL87**

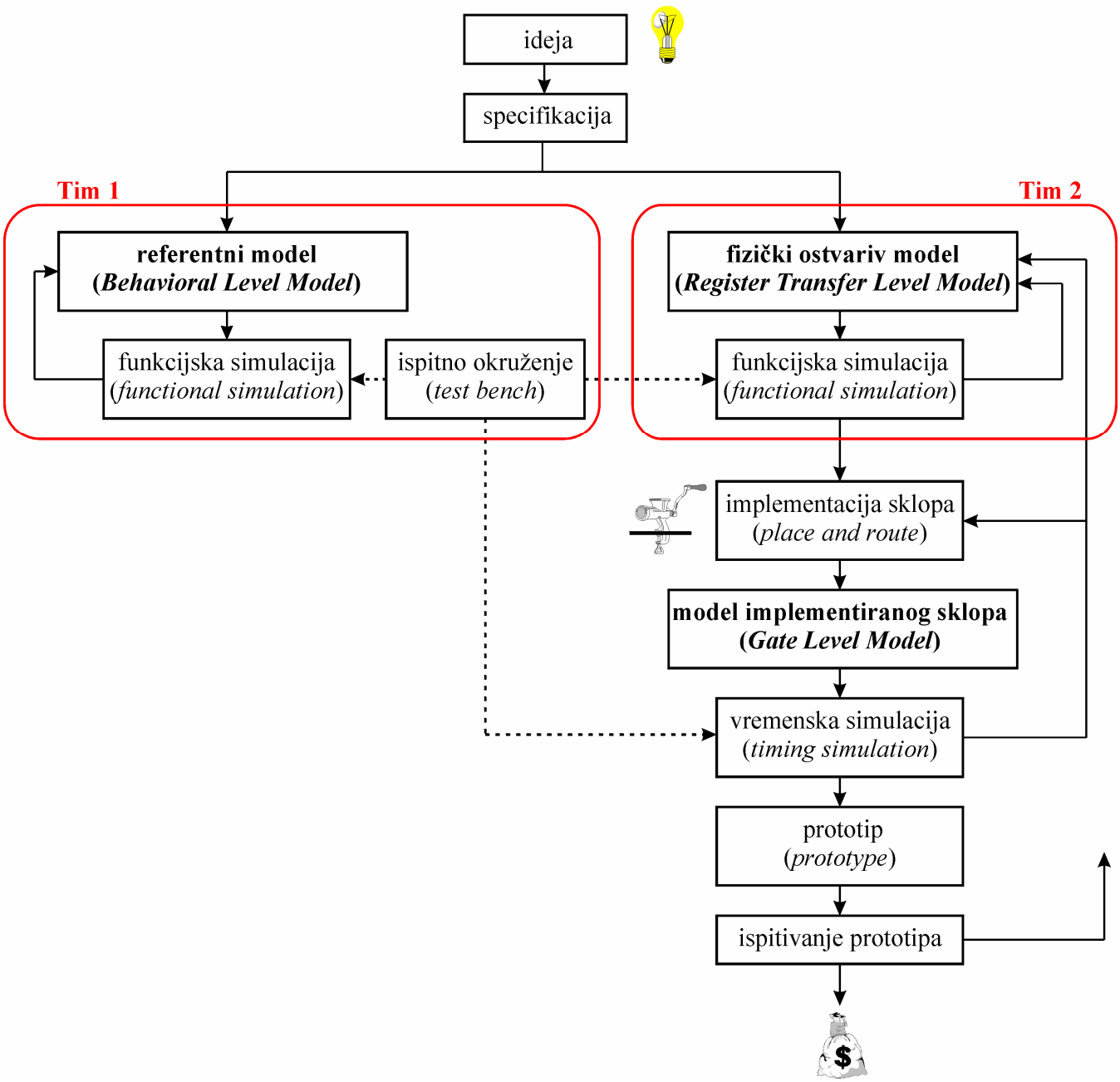
1988. - USA mil. std. 455 - traži od proizvođača ASIC komponenata VHDL

1993. - IEEE Standard 1174-1993 => **VHDL93**

itd.

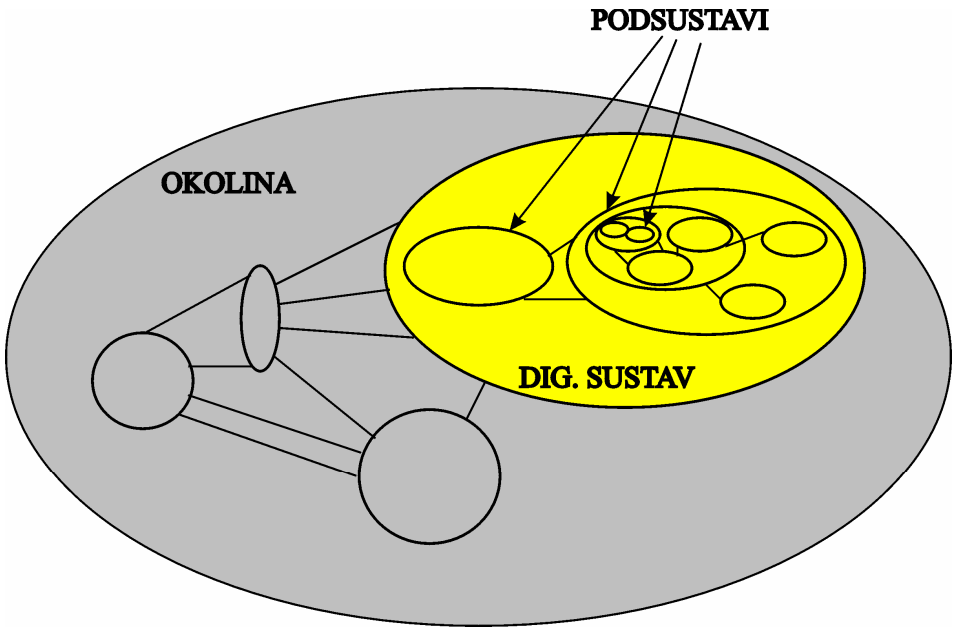
RAZVOJ DIGITALNIH SUSTAVA POMOĆU JEZIKA VHDL

- ***system levels design***



VHDL - OSNOVNI POJMOVI

Građa digitalnog sustava i njegova veza s okolinom



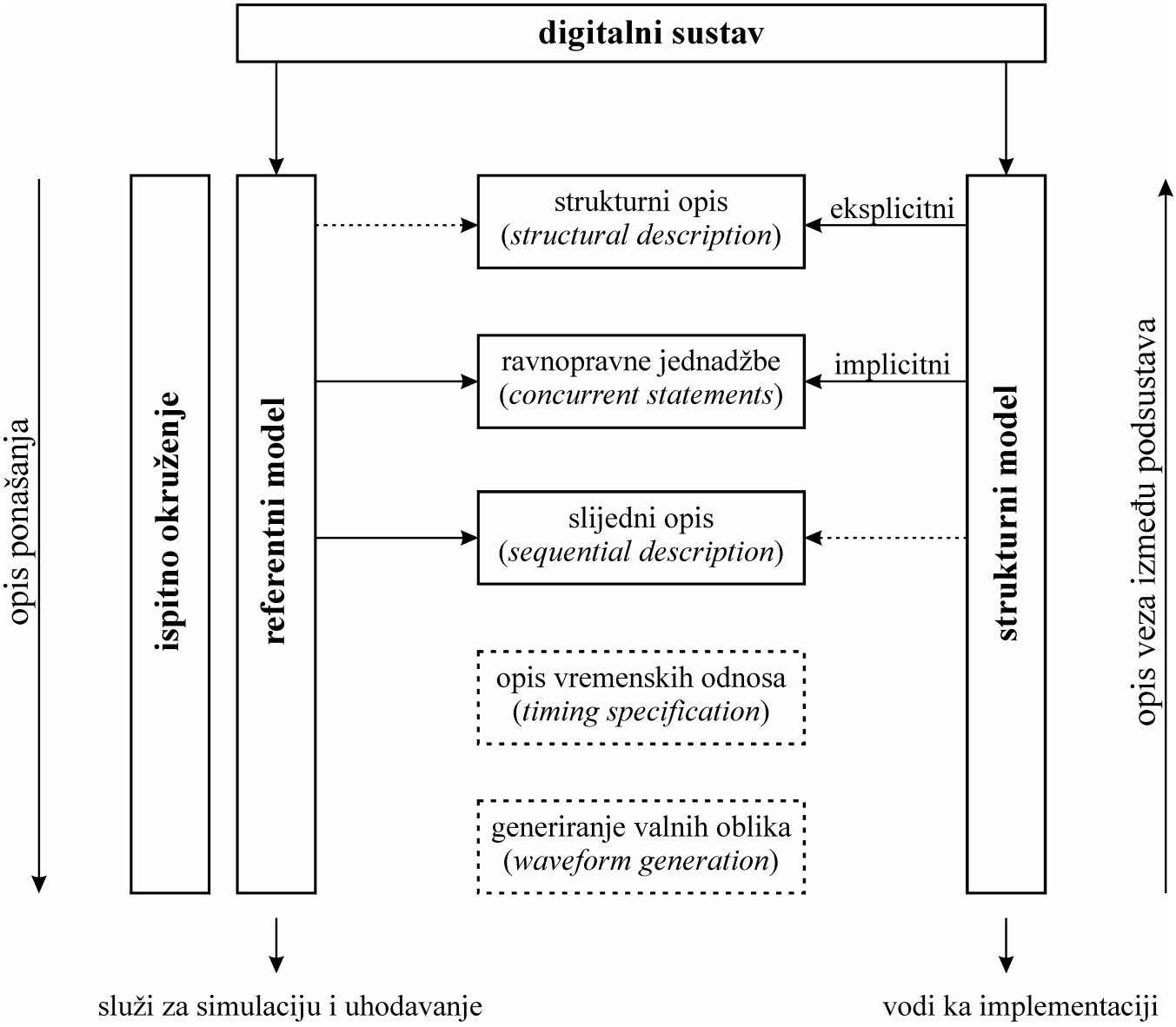
Primjer

- okolina => računalni sustav s mikrokontrolerom (RAM, ROM, izvor takta i dr.)
- digitalni sustav => mikrokontroler
- podsustavi => instrukcijski registar, programsko brojilo, itd.

Osnovni pojmovi

- entitet (*entity*) - abstraktni model sustava odnosno podsustava
- ispitno okruženje (*test bench*) - apstraktni model okoline
- za opis entiteta koriste se slijedeće cjeline (*design units*):
    - deklaracija entiteta (*entity declaration*)
    - arhitektura entiteta (*architecture body*)
    - deklaracija konfiguracije (*configuration declaration*)
    - deklaracija paketa (*package declaration*)
    - sadržaj paketa (*package body*)

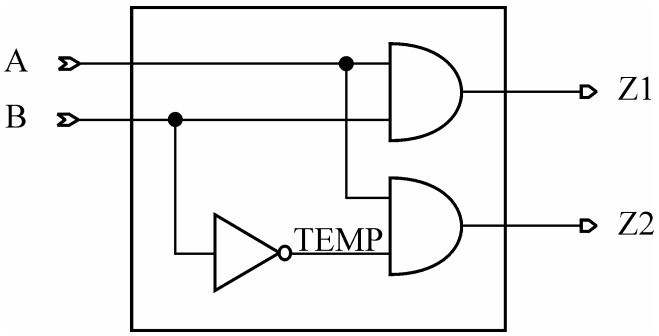
Ilustracija opisa digitalnog sustava



Osnovna građa datoteke koja sadrži VHDL model

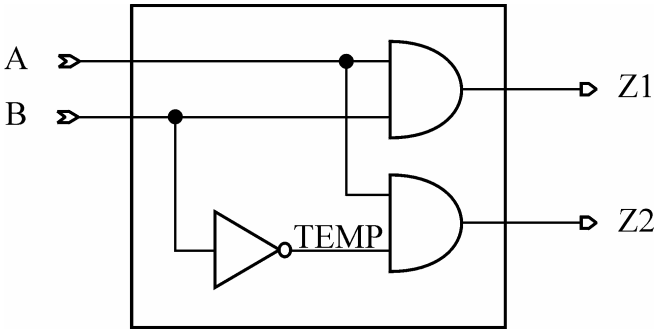
- navođenje korištenih paketa
- deklaracija entiteta
- arhitektura entiteta

Primjer 1.

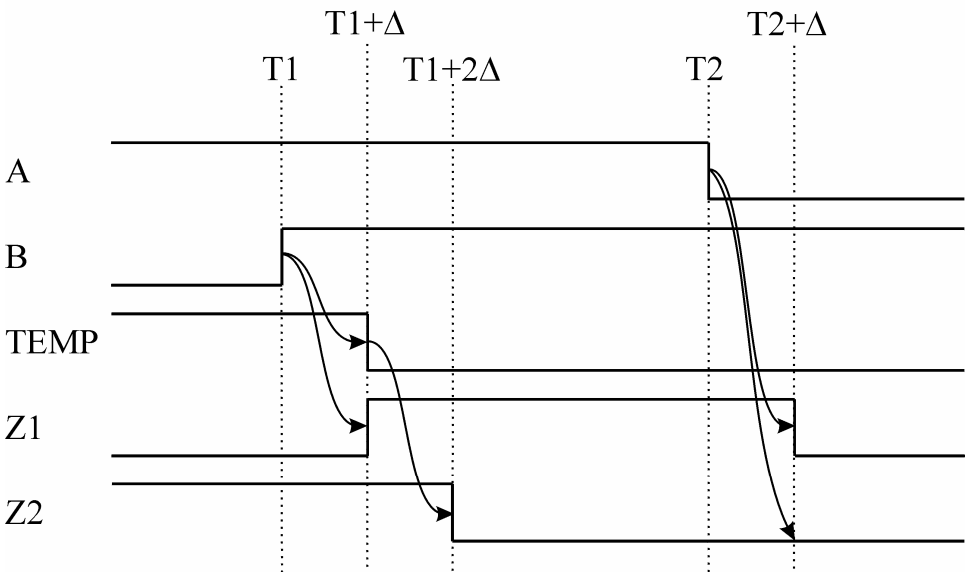



```
-- Deklaracija entiteta.

entity Selector is
    port ( A : in BIT;
           B : in BIT;
           Z1 : out BIT;
           Z2 : out BIT);
end Selector;
```

Primjer 2.

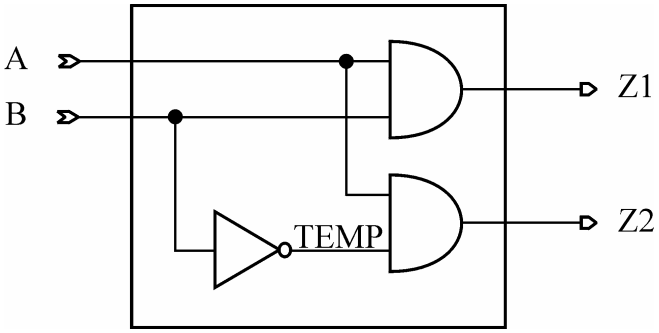



```
-- Primjer arhitekture koja koristi opis entiteta
-- ravnopravnim jednadzbama.

architecture Arh2 of Selector is

   signal TEMP: BIT;

begin

   -- ravnopravne jednadzbe (concurrent statements)

   TEMP <= not B;
   Z1 <= A and B;
   Z2 <= A and TEMP;

end Arh2;
```

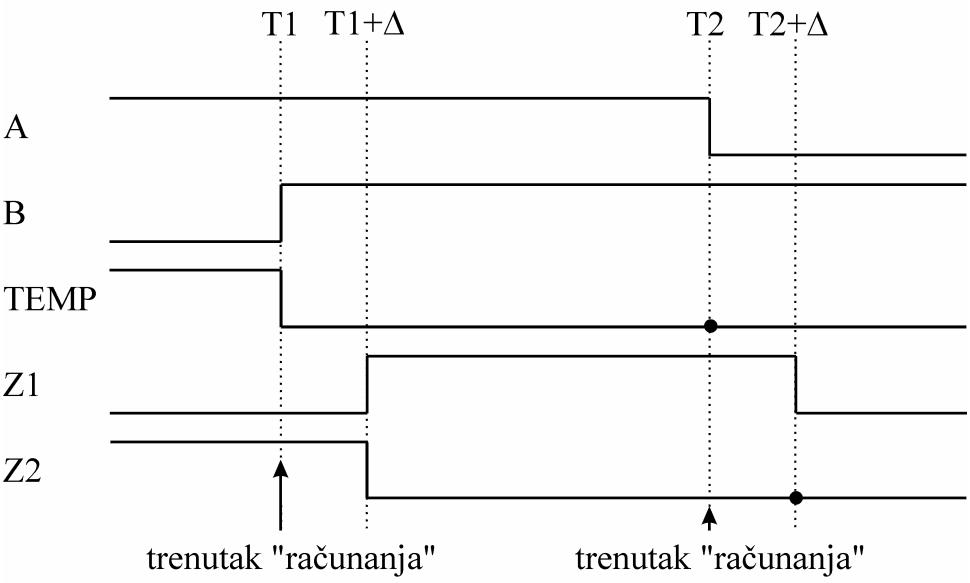


- sve jednadžbe predstavljaju signale
- pojedina jednadžba se "računa" onda kad se promijeni neki argument na njenoj desnoj strani
- izračunata izlazna vrijednost se dodjeljuje nakon vremena Δ (oznaka <=)

Primjer 3.



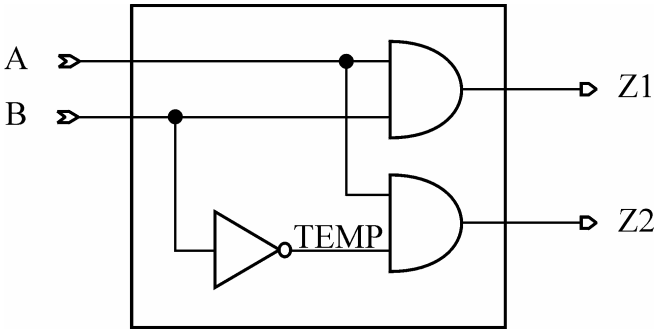

```
-- Primjer arhitekture koja koristi sekvencijalni opis entiteta.

architecture Arh3 of Selector is

begin

   process (A, B) is
      variable TEMP: BIT;   -- deklaracija lokalne varijable
   begin                    -- pocetak procesa
      TEMP := not B;
      Z1 <= A and B;
      Z2 <= A and TEMP;
   end process;             -- kraj procesa

end Arh3;
```



- kad se promijeni bilo koji argument procesa sve jednadžbe se "računaju" u tom trenutku, redom kojim su napisane
- pritom se vrijednosti varijabli mijenjaju u trenutku promjene (:=), a signala nakon vremena Δ (<=)

Primjer 4.

```
-- Primjer arhitekture koja koristi strukturni opis entiteta.

architecture Arh1 of Selector is

   -- Deklaracija komponenata.
   -- Ovdje su deklarirane koristene komponente s opisom njihovih
   -- sucelja.

   component INV
      port (A: in BIT; Z: out BIT);
   end component;

   component AND2
      port (X,Y: in BIT; Z: out BIT);
   end component;

   -- Specifikacija konfiguracije.
   -- Za gore deklarirane komponente koriste se slijedeci entiteti:

   for I1: INV
      use entity WORK.INV;

   for A1,A2 : AND2
      use entity WORK.AND2;

   signal TEMP: BIT;

begin

   -- Formiranje liste spajanja (net-list):

   I1: INV port map (B, TEMP);
   A1: AND2 port map (A, B, Z1);
   A2: AND2 port map (A, TEMP, Z2);

end Arh1;
```



ELEMENTI JEZIKA VHDL

Poznavanje jezika pokriva

- formalni dio
  - sučelje prema čovjeku
    - skup znakova (*character set*)
    - elementi jezika (*lexical elements*);
      identifikatori, rezerviranje riječi itd.
    - sintaksa (*syntax*); gramatika jezika
  - semantika (*semantics*);
    značenje pojedinih dijelova koda, tj. "ono što kod radi"
- primjena

  => stvarno "kreativno" pisanje (modeliranje)

U daljnjem tekstu neće biti rađena ovako stroga podjela

=> primjeri

=> **VHDL93**

=> formalni aspekti i primjena bit će obrađivani paralelno

VHDL87 - nešto jednostavniji i danas se manje koristi

Komentari

počinje znakom "--" i završava krajem reda

Primjer 5.

```
-- ovo je komentar
ovo nije komentar
-- ovo je opet komentar
```

Identifikatori

Imena (identifikatori) sastoje se od

- slova (a-z, A-Z)
- brojeva, tj. znamenki (0-9)
- znaka "_"

Pri tome mora biti zadovoljeno

- ime mora početi slovom (a ne brojem ili znakom "_")
- ime ne smije završiti znakom "_"
- dva znaka "_" ne smiju doći u imenu neposredno jedan iza drugog
- kao ime ne smije biti upotrijebljena neka od rezerviranih riječi

Velika i mala slova se NE razlikuju.

Primjer 6.

```
-- pravilno nazvani identifikatori
RESET   -- isto sto i reset ili ReSeT
A
Ulaz14
privremena_memorija
```

Primjer 7.

```
-- NEpravilno nazvani identifikatori
RESET#
A_
2Ulaz14
privremena__memorija
```

Rezervirane riječi - ima ih 97 (VHDL93)

abs, access, after, alias, all, and, architecture, array, assert, attribute
begin, block, body, buffer, bus
case, component, configuration, constant
disconnect, downto
else, elsif, end, entity, exit
file, for, function
generate, generic, group, guarded
if, impure, in, inertial, inout, is
label, library, linkage, literal, loop
map, mod
nand, new, next, nor, not, null
of, on, open, or, others, out
package, port, postponed, procedure, process, pure
range, record, register, reject, rem, report, return, rol, ror
select, severity, signal, shared, sla, sll, sra, srl, subtype
then, to, transport, type
unaffected, units, until, use
variable
wait, when, while, with
xnor, xor

(Riječi koje nisu podcrtane uključene su u VHDL87)

Objekti

- objekt (*object*) - svaka jedinica opisana tipom i vrijednošću
- **postoje 4 klase (*class*) objekata**
    - konstante (*constants*)
    - varijable (*variables*)
    - signali (*signals*)
    - datoteke (*files*)

Konstante

Glavne značajke konstanti:

- u toku simulacije NE može se promijeniti vrijednost
- vrijednost se dodjeljuje prije početka simulacije

Deklaracija konstante:

```
-- deklaracija jedne konstante
constant ImeKonstante: TipKonstante:= VrijednostKonstante;


-- deklaracija vise konstanti istog tipa i vrijednosti
constant ImeK1, ImeK2, ...: TipKonstante:= VrijednostKonstanti;
```

Primjer 8.

```
-- deklaracija pojedinacnih konstante
constant ADRESA: integer:= 256;
constant EUR2DEM: real:= 1.955830;
constant Pauza: time:= 12 ms;  -- obavezno razmak ispred ms

-- deklaracija vise konstanti istog tipa i vrijednosti
constant F1, F2: integer:= 2*ADRESA; -- vrijednost je izraz

-- deklaracija bez dodjeljivanja vrijednosti
constant REGISTAR1: integer; -- vidi slijedeca poglavlja
```

Varijable

Glavne značajke varijabli:

- u toku simulacije može se promijeniti vrijednost
- vrijednost se može dodijeliti i prije početka simulacije

Deklaracija varijable:

```
-- deklaracija jedne varijable
variable ImeVarijable: TipVarijable:= VrijednostVarijabli;

-- deklaracija vise varijabli istog tipa i pocetne vrijednosti
variable ImeV1, ImeV2,...: TipVarijable:= VrijednostVarijabli;
```

Primjer 9.

```
-- deklaracija pojedinacnih varijabli
variable BROJAC: integer:= 0;
variable Izlaz: real:= 123.4;
variable POCETAK_MJERENJA: time:= 0 us;

-- deklaracija vise varijabli istog tipa i vrijednosti
variable Suma1, Suma2, Suma3: integer:= 0;

-- deklaracija bez dodjeljivanja pocetne vrijednosti
variable Pribrojnik1: integer;
variable C1,C2: bit;
```

- ako se izostavi **`VrijednostVarijable`**

  $\Rightarrow$ uzima se predefinirana vrijednost definirana u deklaraciji tipa

Promjena vrijednosti varijabli (*variable assignment*):

```
-- opci slucaj; labela sluzi za oznacavanje ove linije koda
labela: ImeVarijable:= VrijednostVarijable;

-- uobicajeno; bez labele
ImeVarijable:= VrijednostVarijable;
```

- **`VrijednostVarijable`** je izraz

Primjer 10.

```
-- dodjela vrijednosti varijablama
pocetak_primjera: Pribrojnik1 := 6;
                  Suma1 := Pribrojnik1-45;
                  C1 := not C2;
```

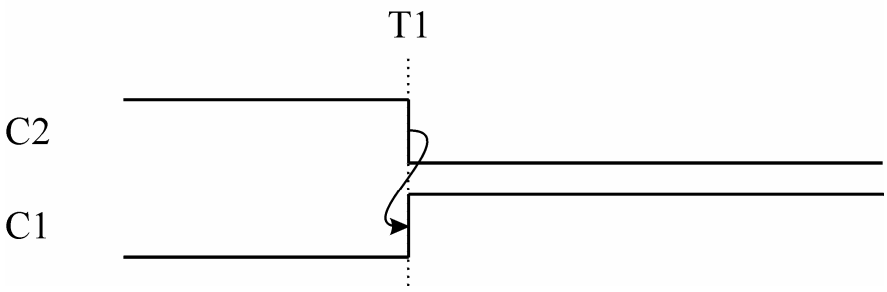
Uočiti

- za dodjeljivanje vrijednosti varijabli koristi se znak **`:=`**

Što će učiniti simulator?

- vrijednost varijable promijenit će se trenutno

```
C1 := not C2;
```



- simulator pamti samo trenutnu vrijednost varijable

Signali

Glavne značajke signala:

- sadrži sva prošla stanja i trenutno stanje
- predstavlja valni oblik
- dodjeljuje mu se **buduća** vrijednost
- tipična primjena ⇒ "žice u uređaju"

Deklaracija signala:

```
-- deklaracija jednog signala
signal ImeSignala: TipSignala:= PocetnoStanje;

-- deklaracija vise signala istog tipa i pocetne vrijednosti
signal Sig1, Sig2, Sig3: TipSignala:= PocetnoStanje;
```

Primjer 11.

```
-- deklaracija pojedinacnih signala
signal GlavniTakt: bit:= '0';
signal IZLAZ1: std_logic:= '1';
signal Operand1: integer:= 100;

-- deklaracija vise signala istog tipa i vrijednosti
signal Rx, Tx: bit:= '0';

-- deklaracija bez dodjeljivanja pocetne vrijednosti
signal Rx, Tx, S1, S2, S3, S4: bit;
signal GlavniTakt: bit;
```

- ako se izostavi **`PocetnoStanje`**

  ⇒ uzima se predefinirana vrijednost definirana u deklaraciji tipa

Promjena stanja signala (*signal assignment*):

```
-- opci slucaj; labela sluzi za oznacavanje ove linije koda
labela: ImeSignala <= ValniOblik;

-- uobicajeno; bez labele
ImeSignala <= ValniOblik;
```

- izraz određuje valni oblik signala nakon trenutka njegovog izvršavanja

Primjer 12.

```
-- dodjela stanja signalima
pocetak_primjera: S2 <= not S1; -- podrazumijeva se after delta
                  S3 <= S1 after 1 us;
                  S4 <= '1', '0' after 1 us, '1' after 3 us;
```

Uočiti

- za dodjeljivanje stanja signalu koristi se znak **<=**
- za dodjeljivanje početnog stanja signalu koristi se znak **:=**

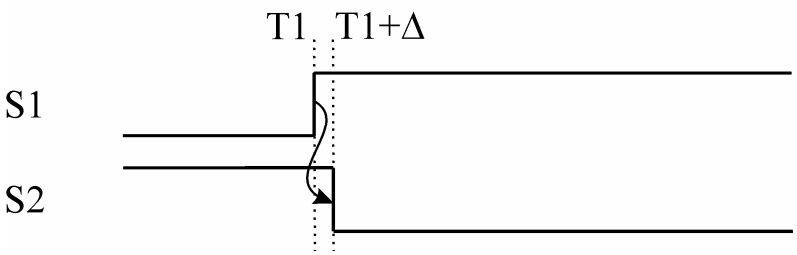
Što će učiniti simulator?

- početno stanje bit će dodijeljeno prije početka simulacije
- simulator pamti cijelu povijest signala
- vrijednost signala računa se u nekom određenom trenutku T1
- izrazom je određena "budućnost" signala nakon trenutka T1

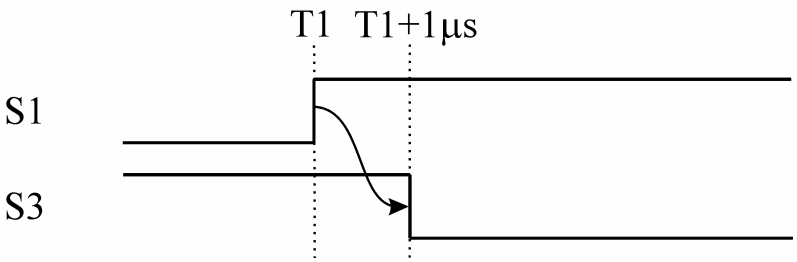
Primjer 13.

Slijedeći izrazi definiraju valni oblik signala nakon trenutka T1.
(Stanje prije trenutka T1 je pretpostavljeno.)

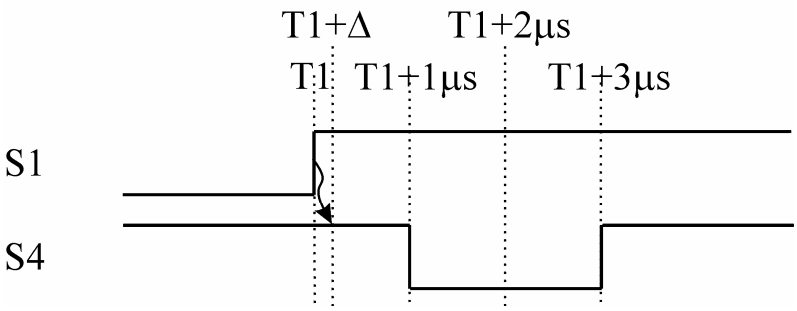
```
S2 <= not S1; -- izraz se racuna u T1, a S2 se postavlja u T1+Δ
```




```
S3 <= not S1 after 1 us; -- "not S1" se racuna u T1,
                         -- a S3 se postavlja u T1+1us
```




```
S4 <= not S1, '0' after '1' us, '1' after 3 us;
```



Datoteke

- veza između VHDL dizajna i programske okoline
- služi za učitavanje i spremanje ispitnih vektora i sl.

Deklaracija datoteke:

```
file LogickoImeDatoteke: TipDatoteke is "FizickoImeDatoteke";
```

- **LogickoImeDatoteke** je ime pod kojim se datoteka prepoznaje unutar VHDL dizajna.
- **FizickoImeDatoteke** je ime datoteke pod kojim ju prepoznaje operacijski sustav.
- **TipDatoteke** opisuje tip objekata koji se nalaze u datoteci.

Primjer 14.

```
-- deklaracija datoteke
file IzlazniPodaci: CjelobrojniTip is "d:\users\proj4\rez1.int";
```

- više o datotekama ⇒ vidi daljnji tekst

Zajedničko dosad spomenutim objektima

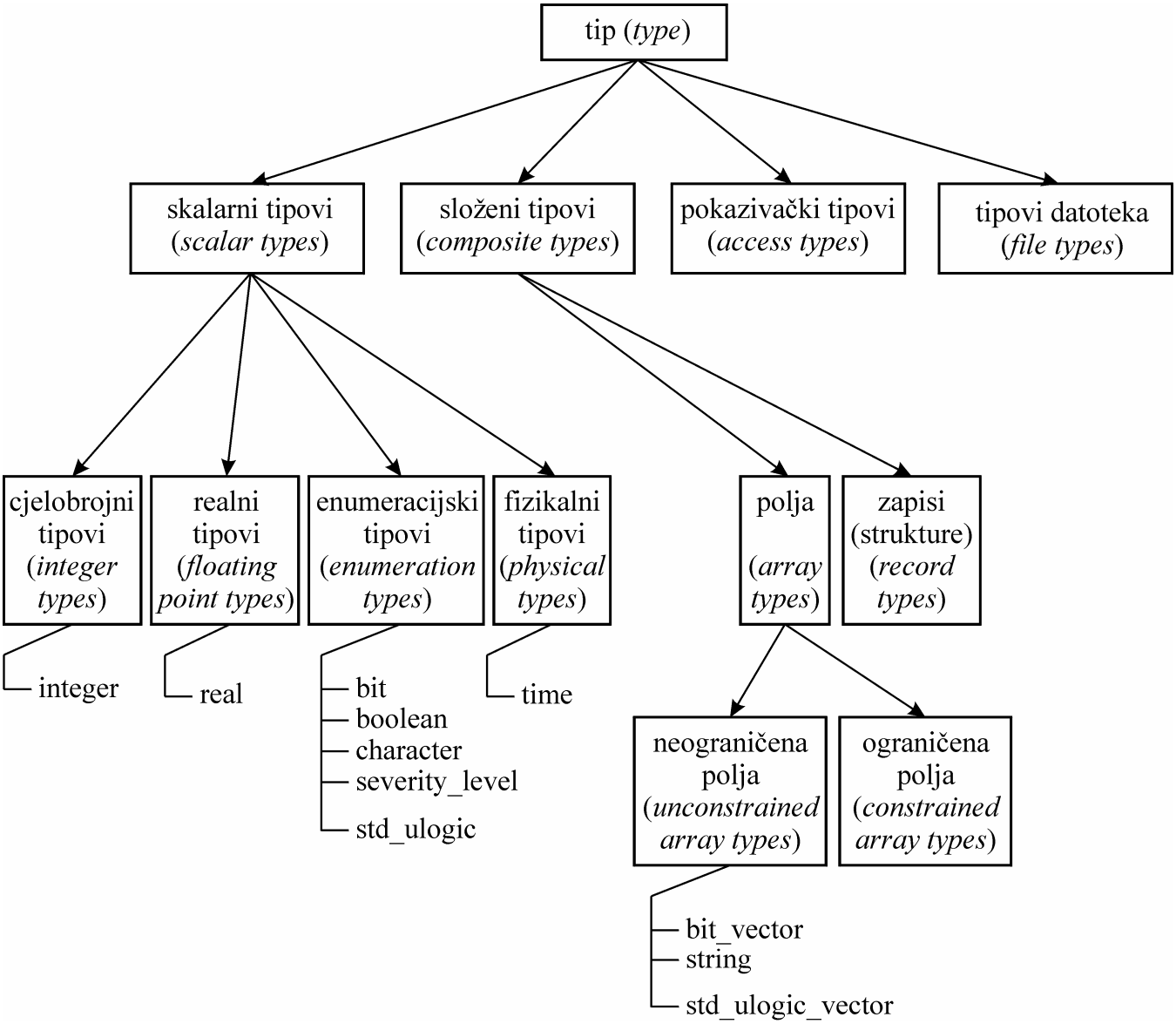
tip

⇒ operacije s pojedinim tipovima

Tipovi objekata

- tip (*type*) - definiraju skup vrijednosti pojedinog objekta

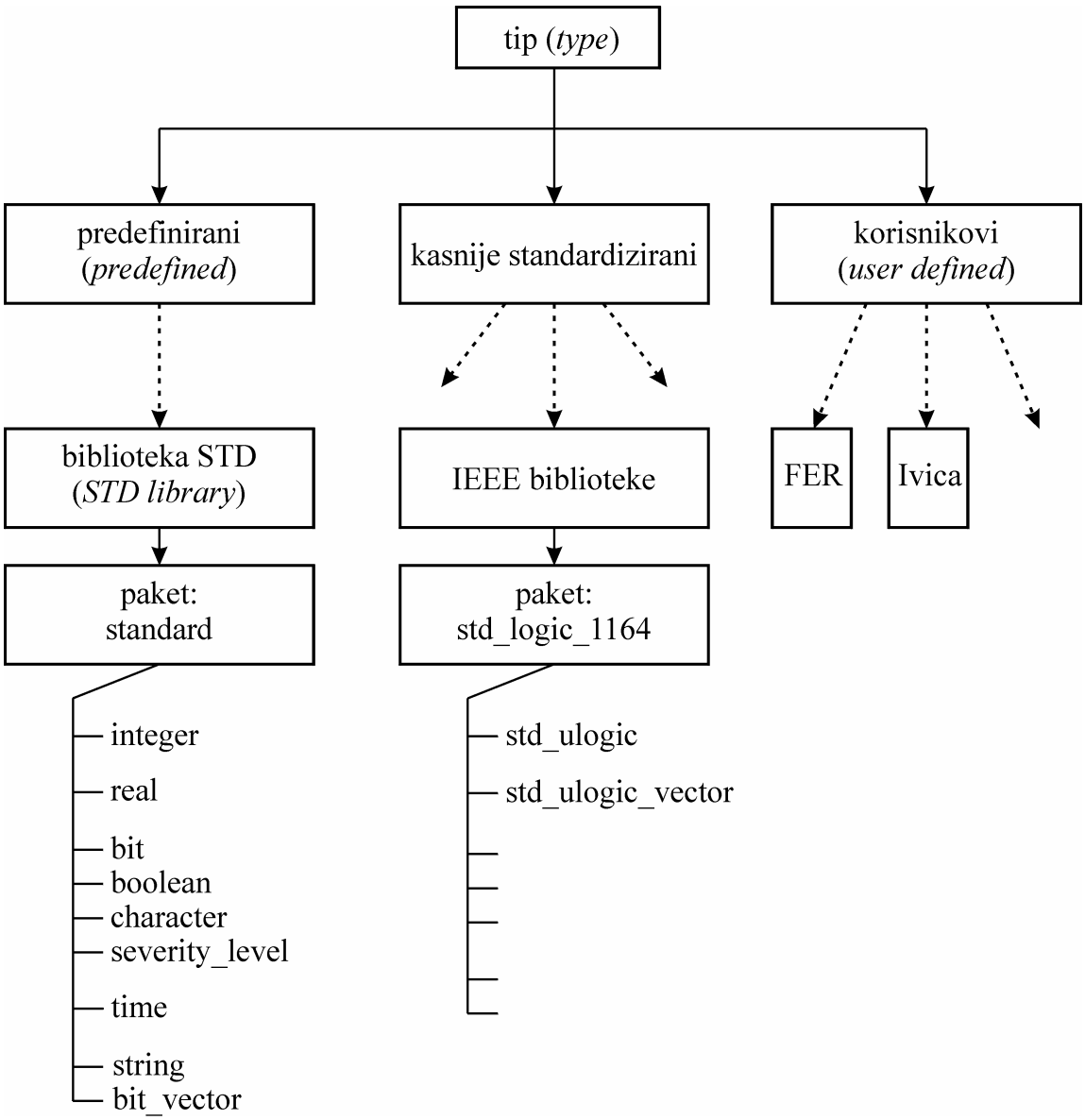
Pregled tipova



Uočiti:

- integer, real, bit, boolean itd. su imena tipova
- razlikovati *integer types* od integer
  (integer je samo jedan tip koji pripada grupi cjelobrojnih tipova)

Standardni tipovi



Deklaracija novog tipa:

```
type ImeTipa is DefinicijaTipa;
```

- **`ImeTipa`** koristi se u deklaraciji objekta ⇒ vidi prethodni tekst
  (**`TipVarijable`**, **`TipKonstante`**, itd.)
- **`DefinicijaTipa`** opisuje dotični tip ⇒ vidi daljnji tekst

Skalarni tipovi objekata

Cjelobrojni tipovi

Deklaracija cjelobrojnog tipa:

```
type ImeCjTipa is range DonjaGranica to GornjaGranica;
type ImeCjTipa is range GornjaGranica downto DonjaGranica;
```

- **DonjaGranica** i **GornjaGranica** su CIJELI brojevi (izrazi)
- vrijednost koja slijedi nakon **range** je predefinirana vrijednost koja se uzima kao inicijalno stanje varijabli, konstanti i signala (vidi deklaracije konstanti i varijabli)

Primjer 15.

```
-- deklaracija novih tipova

type BrojBodova is range 0 to 25;          -- rastuci interval
type Ocjena is range 1 to 5;               -- rastuci interval
type GodinaRodenja is range 1900 to 2100; -- rastuci interval

type Odbrojavanje is range 10 downto 0;    -- padajuci interval
```

Primjer 16.

```
-- ISPRAVNA upotreba deklariranih tipova

type Ocjena is range 1 to 5;               -- rastuci interval

constant PadNaIspitu: Ocjena:= 1;
constant Izvrstan: Ocjena:= 5;

-- pocetna vrijednost ovih varijabli iznosi 1, jer nema ":=..."
variable SeminarPeric: Ocjena;
variable SeminarPetrovic: Ocjena;

-- ispravno dodjeljivanje vrijednosti varijabli
SeminarPeric:=Izvrstan; -- isto kao da pise SeminarPeric:=5;
SeminarPetrovic:=PadNaIspitu;
```

Primjer 17.

```
-- NEISPRAVNA upotreba deklariranih tipova

type BrojBodova is range 0 to 25;
type Ocjena is range 1 to 5;

constant Dovoljan: Ocjena:= 2;

variable SeminarPeric: Ocjena;
variable BodovaPeric: BrojBodova:=0;

-- NEISPRAVNO jer je 26 izvan intervala tipa BrojBodova
BodovaPeric:=26;

-- ISPRAVNO jer je 2 cjelobrojna vrijednost unutar intervala
-- tipa BrojBodova
BodovaPeric:=2;

-- NEISPRAVNO jer je Dovoljan konstanta "pogresnog" tipa
BodovaPeric:=Dovoljan;
```

Predefiniran cjelobrojni tip

```
-- ovaj tip deklariran je u bilioteci STD, paketu STANDARD
type integer is range -2147483648 to 2147483647;
```

Uočiti

- predefinirana vrijednost tipa (*default value*) je broj koji slijedi nakon **`range`**
- dva različito deklarirana tipa se NE mogu "miješati" iako su oba npr. cjelobrojna
- **`integer`** je samo jedan naziv cjelobrojnog tipa koji je predefiniran !

Operacije nad cjelobrojnim tipovima

OPERATOR	PRIMJER	OPIS
+	x+y	2+3=5
-	x-y	2-3=-1
*	x*y	2*3=6
/	x/y	cjelobrojno dijeljenje zaokružuje se prema 0 12/6=2, 14/6=2, (-14)/6=-2, 14/(-6)=-2
**	x**y	2**4=16, 10**2=100 (y mora biti pozitivan, tipa integer)
mod	x mod y	x mod y = x - y*floor(x/y) , y ≠ 0 8 mod 3 = 2, 8 mod (-3) = -1 (-8) mod 3 = 1 (-8) mod (-3) = -2 (predznak rezultata je predznak od y)
rem	x rem y	x rem y = x - y*fix(x/y), y ≠ 0 8 rem 3 = 2 8 rem (-3) = 2 (-8) rem 3 = -2 (-8) rem (-3) = -2 (predznak rezultata je predznak od x)
abs	abs(x)	abs(4)=4, abs(-6)=6, abs(0)=0

- operandi i rezultat moraju biti ISTOG tipa
- **rem i mod** daju isti rezultat ako je sign(x)=sign(y)
- kod potenciranja, **integer** označava predefinirani, a ne proizvoljan cjelobrojni tip

Realni tipovi

- predstavljaju aproksimacije realnih brojeva

Deklaracija realnog tipa:

```
type ImeReTipa is range DonjaGranica to GornjaGranica;
type ImeReTipa is range GornjaGranica downto DonjaGranica;
```

- **DonjaGranica** i **GornjaGranica** su REALNI brojevi (izrazi)
- vrijednost koja slijedi nakon **range** je predefinirana vrijednost koja se uzima kao inicijalno stanje varijabli, konstanti i signala (vidi deklaracije konstanti i varijabli)

Primjer 18.

```
-- deklaracija novih tipova

type SrednjaOcjena is range 2.0 to 5.0;      -- rastuci interval
type OcitanjePoDanu is range 0.0 to 5000.0; -- rastuci interval

type RV is range 10.0 downto -10.0;          -- padajuci interval
```

Primjer 19.

```
-- ISPRAVNA upotreba deklariranih tipova

type SrednjaOcjena is range 2.0 to 5.0;

constant StariProsjek: SrednjaOcjena:= 4.99;

-- pocetna vrijednost varijabli iznosi 2.0
variable NoviProsjek, Razlika: SrednjaOcjena;

-- ispravno dodjeljivanje vrijednosti varijabli
Razlika:=NoviProsjek-StariProsjek;
StariProsjek:=NoviProsjek;
```

Primjer 20.

```
-- NEISPRAVNA upotreba deklariranih tipova

type SrednjaOcjena is range 2.0 to 5.0;
type SrednjaOcjena1 is range 2.0 to 5.0;
type Ocjena is range 1 to 5;

variable Var1: SrednjaOcjena;
variable Var2: SrednjaOcjena1;
variable Var3: Ocjena;

-- NEISPRAVNO dodjeljivanje vrijednosti varijablama
Var1:=Var2;
Var1:=Var3;
```

Predefiniran realni tip

```
-- ovaj tip deklariran je u bilioteci STD, paketu STANDARD
type real is range -1.0E308 to 1.0E308;
```

Uočiti

- predefinirana vrijednost tipa (*default value*) je broj koji slijedi nakon **`range`**
- dva različito deklarirana tipa se NE mogu "miješati"
- **`real`** je samo jedan naziv realnog tipa, koji je predefiniran !

Operacije nad realnim tipovima

OPERATOR	PRIMJER	OPIS
+	x+y	2.0+3.0=5.0
-	x-y	2.0-3.0=-1.0
*	x*y	2.0*3.0=6.0
/	x/y	dijeljenje 12.0/5.0=2.4, (-12.0)/5.0=-2.4
**	x**y	2.3**4=23.4256, 10.0**2=100.0 (y mora biti tipa integer)
abs	abs(x)	abs(4.23)=4.23 abs(-6.8)=6.8 abs(0.0)=0.0

- operandi i rezultat moraju biti ISTOG tipa
- kod potenciranja, **integer** označava predefinirani, a ne proizvoljan cjelobrojni tip

Enumeracijski tipovi

- služi za opis signala simboličkim imenima

Deklaracija enumeracijskog tipa:

```
type ImeEnTipa is (Iden1, Iden2, Iden3, ...);
                   -- Iden oznacava identifikator (ime)
type ImeEnTipa is ('Kar1', 'Kar2', 'Kar3', ...);
                   -- Kar oznacava karakter
```

- prva vrijednost u zagradi je predefinirana vrijednost koja se uzima kao inicijalno stanje varijabli, konstanti i signala

Primjer 21.

```
-- deklaracija novih tipova

-- dani u tjednu
type Dani is (PON, UTO, SRI, CET, PET, SUB, NED);

-- heksadecimalne znamenke
type HexaBroj is ('0','1','2','3','4','5','6','7','8',
                  '9','A','B','C','D','E','F');
```

Primjer 22.

```
-- ISPRAVNA upotreba deklariranih tipova

type Dani is (PON, UTO, SRI, CET, PET, SUB, NED);
type HexaBroj is ('0','1','2','3','4','5','6','7','8',
                  '9','A','B','C','D','E','F');

variable Praznik: Dani:= UTO;
variable Indikator: HexaBroj:='0';

-- ispravno dodjeljivanje vrijednosti varijabli
Praznik := NED;    -- NEMA navodnika
Indikator := '9'; -- IMA navodnika
```

Predefinirani enumeracijski tipovi

```
-- ovi tipovi deklarirani su u bilioteci STD, paketu STANDARD

type bit is ('0', '1');

type boolean is (false,true);

type character is (
   nul, soh, stx, etx, eot, enq, ack, bel,
   bs,  ht,  lf,  vt,  ff,  cr,  so,  si,
   dle, dc1, dc2, dc3, dc4, nak, syn, etb,
   can, em,  sub, esc, fsp, gsp, rsp, usp,

   ' ', '!', '"', '#', '$', '%', '&', ''',
   '(', ')', '*', '+', ',', '-', '.', '/',
   '0', '1', '2', '3', '4', '5', '6', '7',
   '8', '9', ':', ';', '<', '=', '>', '?',

   '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
   'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
   'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
   'X', 'Y', 'Z', '[', '\', ']', '^', '_',

   '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
   'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
   'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
   'x', 'y', 'z', '{', '|', '}', '~', del,

   -- u nekim standardima ovdje postoji i nastavak, tj.
   -- karakteri ciji je ASCII kod veci od 127 );

type severity_level is (note, warning, error, failure);
```

Uočiti

- predefinirana vrijednost tipa (*default value*) je prva vrijednost u zagradi
- dva različito deklarirana tipa se NE mogu "miješati"
- predefinirani tipovi su: **`bit, boolean, character, severity_level`**

Ostali standardni enumeracijski tipovi

- često korišten paket, kompatibilan sa IEEE standardom 1164

Primjer 23.

- `std_ulogic` ⇒ modelira "stvarni" digitalni signal
- nužan je za pisanje fizički ostvarivog modela (*RTL model*)

```
-- ovaj tip deklariran je u bilioteci IEEE,
-- paketu std_logic_1164

type std_ulogic is ( 'U',  -- Uninitialized
                     'X',  -- Forcing  Unknown
                     '0',  -- Forcing  0
                     '1',  -- Forcing  1
                     'Z',  -- High Impedance
                     'W',  -- Weak     Unknown
                     'L',  -- Weak     0
                     'H',  -- Weak     1
                     '-'   -- Don't care );
```

Operacije s enumeracijskim tipovima

Relacije

OPERATOR	PRIMJER	OPIS
=	x=y	(jednako) 2 = 5 Praznik = NED
/=	x/=y	(različito) 2 /= 5 Indikator /= 'D'
<	x<y	2 < 5 character('b') < character('B')
<=	x<=y	2 <= 5 Praznik <= SUB
>	x>y	2 > 5 Praznik > NED
>=	x>=y	2 >= 5 Praznik >= PON

- operandi mogu biti **svi** (istovrsni) tipovi osim datoteka
- svi operatori u izlazu moraju biti istog tipa
- izlaz je uvijek tipa boolean, tj. false ili true

Logičke operacije

OPERATOR	PRIMJER	OPIS
`and`	`x and y`	`'1' and '0' = '0',` `true and false = false`
`or`	`x or y`	`'1' or '0' = '1',` `true and false = true`
`nand`	`x nand y`	`'1' nand '0' = '1',` `true nand false = true`
`nor`	`x nor y`	`'1' nor '0' = '0',` `true nor false = false`
`xor`	`x xor y`	`'1' xor '1' = '0',` `true xor true = false`
`xnor`	`x xnor y`	`'1' xnor '1' = '1',` `true xor true = true`
`not`	`not y`	`not '0' = '1',` `not false = true`

- operandi su tipa **boolean** ili **bit**
- svi operatori u izrazu moraju biti istog tipa

Primjer 24.

```
-- izrazi mogu biti slozeni

variable a, b, c: bit;
variable d: boolean;

d := (a and b) /= c;
```

Fizikalni tipovi

- služi za predstavljanje stvarnih fizikalnih veličina

Deklaracija fizikalnog tipa:

```
type ImeFizTipa is range DonjaGranica to GornjaGranica
                -- range GornjaGranica downto DonjaGranica
   units
      MjernaJedinica1;              -- osnovna jedinica
      MjernaJedinica2=Definicija;   -- izvedena jedinica
      -- itd.
   end units ImeFizTipa;
```

- **DonjaGranica** i **GornjaGranica** su brojevi (izrazi)
- vrijednost koja slijedi nakon **range** je predefinirana vrijednost koja se uzima kao inicijalno stanje
- razlučivost jedinice jednaka je osnovnoj jedinici

Primjer 25.

```
-- deklaracija novog tipa

type Napon1 is range -1000.0 to 1000.0
   units
      uV;               -- mikrovolt
      mV = 1000 uV;     -- milivolt
      V = 1000 mV;      -- volt
   end units Napon1;

-- deklaracija varijable

variable NaponNapajanja, U1,U2: Napon1;

-- ISPRAVNO dodjeljivanje vrijednosti varijabli

NaponNapajanja := 5 V;  -- izmedu 5 i V mora biti razmak
U1 := 982 mV;

-- NEISPRAVNO dodjeljivanje vrijednosti varijabli

U2 := 1001 mV;  -- jer je izvan dopustenih granica
```

Primjer 26.

```
-- ilustracija razlucivosti

type Napon2 is range -1e8 to 1e8
   units
      uV;              -- mikrovolt
      mV = 1000 uV;    -- milivolt
      V  = 1000 mV;    -- volt
   end units Napon2;

variable U3: Napon2;

-- zaokruzivanje (odsjecanje) zbog razlucivosti

U3 := 3.456 uV;       -- bit ce zaokruzeno na 3 uV
U3 := 0.123456789 V; -- bit ce zaokruzeno na 0.123456 V
```

Predefiniran fizikalni tip

```
-- ovaj tip deklariran je u bilioteci STD, paketu STANDARD

type time is range -2147483647 to 2147483647;
   units
      fs;
      ps  = 1000 fs;
      ns  = 1000 ps;
      us  = 1000 ns;
      ms  = 1000 us;
      sec = 1000 ms;
      min =   60 sec;
      hr  =   60 min;
   end units time;
```

Uočiti

- predefinirana vrijednost tipa (*default value*) je broj koji slijedi nakon **`range`**
- razlučivost je određena osnovnom jedinicom

Podtipovi

- služe za opis objekata koji imaju točno određen skup vrijednosti
- podtip (*subtype*) izvodi se iz prethodno deklariranog osnovnog tipa (*base type*)

Deklaracija podtipa:

```
subtype ImePodtipa is ImeOsnovnogTipa range DoGr to GoGr;
subtype ImePodtipa is ImeOsnovnogTipa range GoGr downto DoaGr;
```

- **DoGr** i **GoGr** su brojevi (izrazi)
- vrijednost koja slijedi nakon **range** je predefinirana vrijednost koja se uzima kao inicijalno stanje

Primjer 27.

```
-- deklaracija podtipa

subtype HI is integer range 0 to 15;
subtype c4 is integer range 128 to 255;
```

Primjer 28.

```
-- deklaracija osnovnog tipa

type Napon1 is range -1000 to 1000
   units
      uV;               -- mikrovolt
      mV = 1000 uV;     -- milivolt
      V  = 1000 mV;     -- volt
   end units Napon1;

-- deklaracija podtipa

subtype NaponSuma is Napon1 range 0 to 10 mV;

-- deklaracija varijable

variable SumUlaz1: NaponSuma;
              -- s podtipom se radi kao i s osnovnim tipom
```

Paket "STANDARD"

```
-- Ovo je primjer paketa STANDARD. U praksi se moze naici na
-- manje razlike u odnosu na dani kod, ovisno o koristenom
-- standardu

package standard is

   type integer is range -2147483648 to 2147483647;
   type real is range -1.0E308 to 1.0E308;
                      -- preciznost ovisi o softvareu

   type bit is ('0', '1');
   type boolean is (false,true);
   type character is (
      nul, soh, stx, etx, eot, enq, ack, bel,
      bs,  ht,  lf,  vt,  ff,  cr,  so,  si,
      dle, dc1, dc2, dc3, dc4, nak, syn, etb,
      can, em,  sub, esc, fsp, gsp, rsp, usp,

      ' ', '!', '"', '#', '$', '%', '&', ''',
      '(', ')', '*', '+', ',', '-', '.', '/',
      '0', '1', '2', '3', '4', '5', '6', '7',
      '8', '9', ':', ';', '<', '=', '>', '?',

      '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
      'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
      'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
      'X', 'Y', 'Z', '[', '\', ']', '^', '_',

      '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
      'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
      'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
      'x', 'y', 'z', '{', '|', '}', '~', del,

       -- u nekim standardima ovdje postoji i nastavak, tj.
       -- karakteri ciji je ASCII kod veci od 127
       );

   type severity_level is (note, warning, error, failure);
```

```
   type time is range -2147483647 to 2147483647;
      units
         fs;
         ps = 1000 fs;
         ns = 1000 ps;
         us = 1000 ns;
         ms = 1000 us;
         sec = 1000 ms;
         min = 60 sec;
         hr = 60 min;
      end units;

   subtype delay_length is time range 0 fs to time'high;

   impure function now return delay_length;

   subtype natural is integer range 0 to integer'high;
   subtype positive is integer range 1 to integer'high;

   type string is array (positive range <>) of character;
   type bit_vector is array (natural range <>) of bit;

   type file_open_kind is (
      read_mode,
      write_mode,
      append_mode);
   type file_open_status is (
      open_ok,
      status_error,
      name_error,
      mode_error);

   attribute foreign : string;

end standard;
```

Pretvorbe tipova

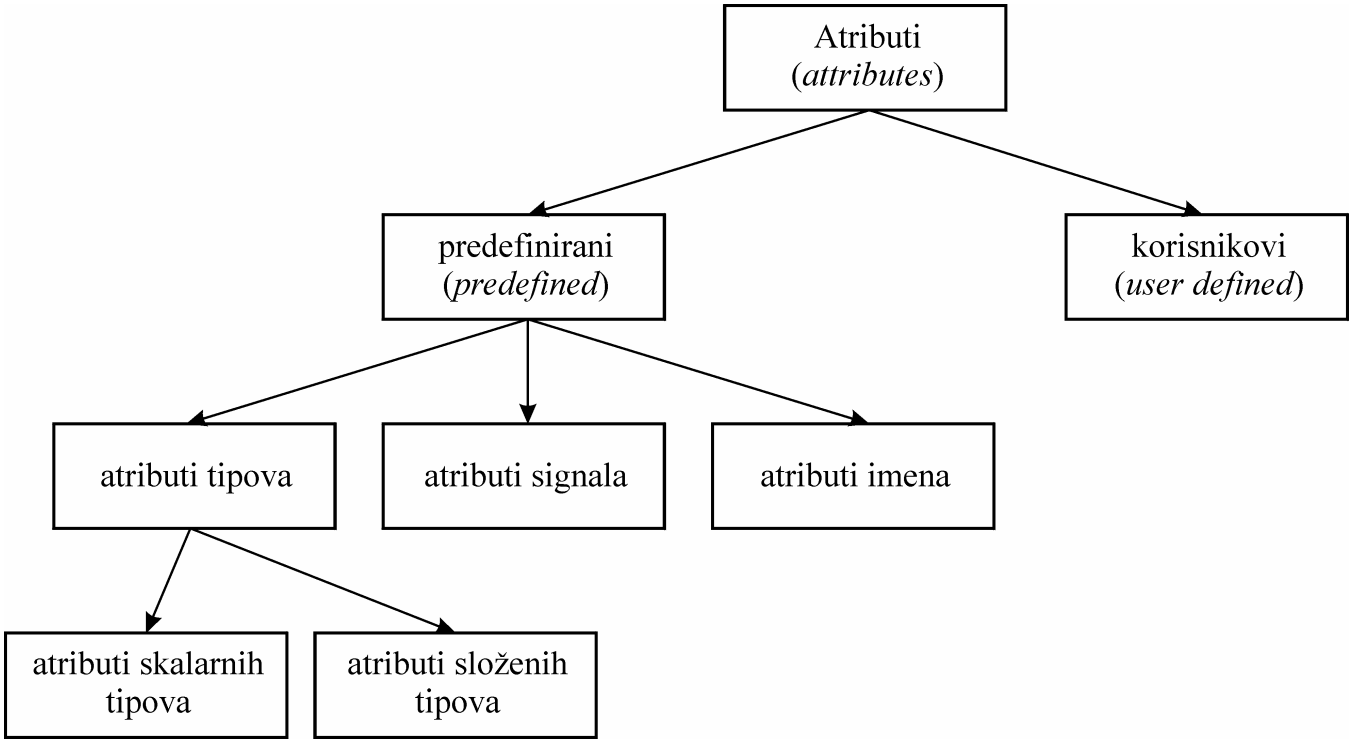
- pretvorba tipa (*type conversion*) koristi se kad se u istom izrazu različiti tipovi objekata
- rade se pretvorbe koje "imaju smisla" ⇒ tipovi moraju biti "slični"

Primjer 29.

```
-- pretvorba real -> integer

integer(3.14); -- dolazi do pogreske zbog zaokruzivanja

-- pretvorba integer -> real

real(725); -- moguca je pogreska uslijed konacne duljine mantise
```

Atributi

Pregled atributa prema objektima na koje se odnose



Atributi tipova

- daju informacije o vrijednostima definiranim u tipu

Primjer 30.

```
-- vrijednosti atributa za dane tipove

type Ocjena is range 1 to 5;

-- vrijednosti atributa
      -- Ocjena'left=1
      -- Ocjena'right=5
      -- Ocjena'low=1
      -- Ocjena'high=5
      -- Ocjena'ascending=true
```

Primjer 31.

```
-- vrijednosti atributa za dane tipove

type Napon1 is range -1000.0 to 1000.0
   units
      uV;              -- mikrovolt
      mV = 1000 uV;    -- milivolt
      V  = 1000 mV;    -- volt
   end units Napon1;

-- vrijednosti atributa
      -- Napon1'left=-1000.0
      -- Napon1'right=1000.0
      -- Napon1'low=-1000.0
      -- Napon1'high=1000.0
      -- Napon1'ascending=true
      -- Napon1'image(2 mV)="2000 uV"
      -- Napon1'value("2 mV")=2000 uV
```

Primjer 32.

```
-- vrijednosti atributa za dane tipove

type SKUP_INSTRUKCIJA is (LOAD,ADD,SUB,MUL,DIV,NOP,STORE);
-- vrijednosti atributa
      -- SKUP_INSTRUKCIJA'pos(ADD)=1
      -- SKUP_INSTRUKCIJA'val(1)=ADD
      -- SKUP_INSTRUKCIJA'succ(ADD)=SUB
      -- SKUP_INSTRUKCIJA'leftof(ADD)=LOAD
      -- SKUP_INSTRUKCIJA'rightof(ADD)=SUB
```

Predefinirani atributi za skalarne tipove objekata

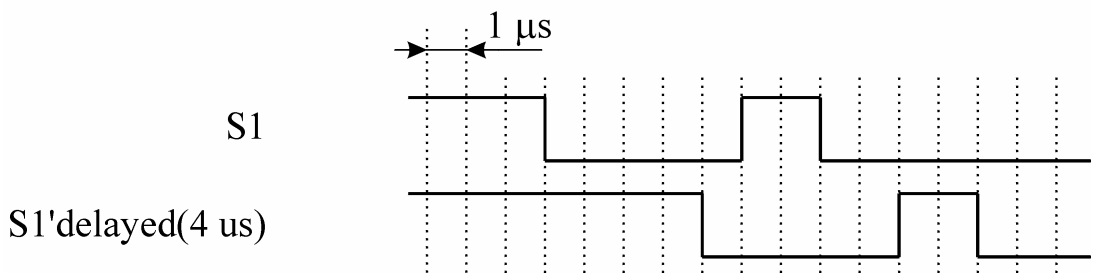
Atribut	Opis	Primjenjiv na tip
`Tip'left`	lijeva vrijednost u definiciji tipa	cjelobrojni realni enumeracijski fizikalni
`Tip'right`	desna vrijednost u definiciji tipa	
`Tip'low`	najmanja vrijednost u definiciji tipa	
`Tip'high`	najveća vrijednost u definiciji tipa	
`Tip'ascending`	**`true`** za rastući interval (**`to`**)	
`Tip'image(x)`	string koji predstavlja **`x`**	
`Tip'value(s)`	vrijednost objekta predstavljena stringom **`s`**	
`Tip'pos(x)`	pozicija vrijednosti **`x`** u definiciji tipa	cjelobrojni enumeracijski fizikalni
`Tip'val(n)`	vrijednosti na **`n`**-toj poziciji	
`Tip'succ(x)`	vrijednosti na prvoj većoj poziciji u odnosu na **`x`**	
`Tip'pred(x)`	vrijednosti na prvoj manjoj poziciji u odnosu na **`x`**	
`Tip'leftof(x)`	vrijednosti na prvoj lijevoj poziciji u odnosu na **`x`**	
`Tip'rightof(x)`	vrijednosti na prvoj desnoj poziciji u odnosu na **`x`**	

Atributi signala

- postoje dva tipa atributa signala
    - implicitni signali
    - vrijednosti određenih parametara signala

Primjer 33.

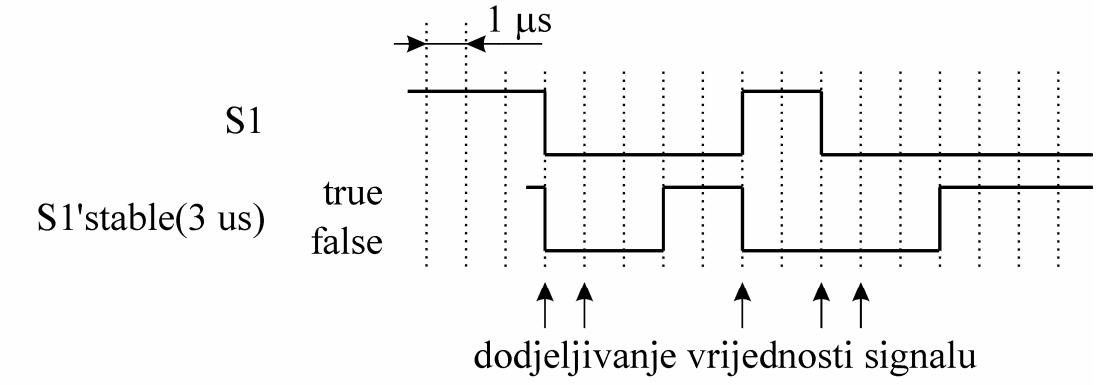
`Signal'delayed(T)`



- **`Signal'delayed(4 us)`** predstavlja signal S1 pomaknut za 4 μs

Primjer 34.

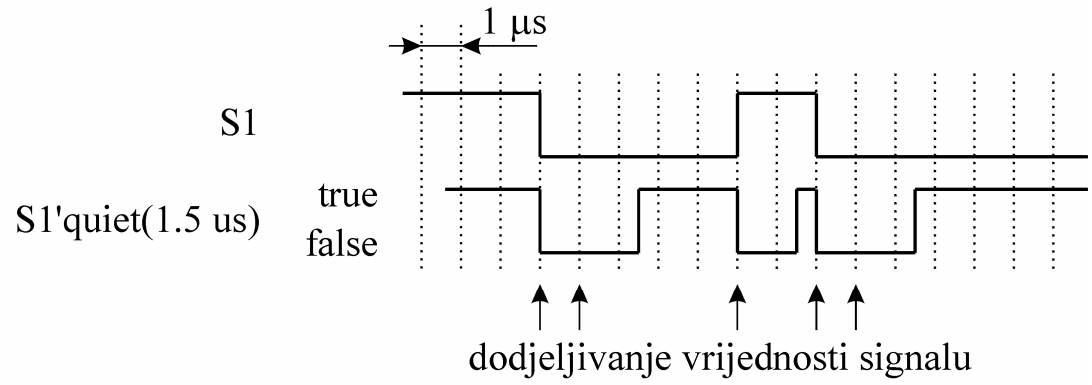
`Signal'stable(T)`



- **`Signal'stable(3 us)`** postaje **`true`** 3 μs nakon brida (promjene) signala S1

Primjer 35.

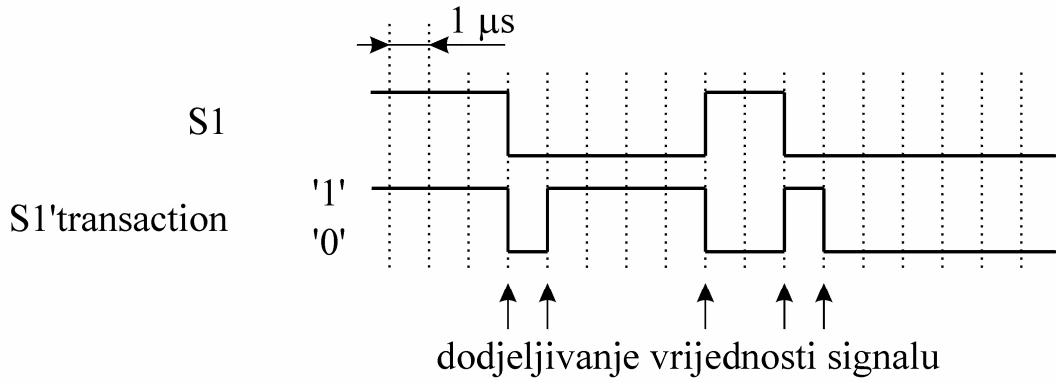
`Signal'quiet(T)`



- **`Signal'quiet(1.5 us)`** postaje **`true`** 1.5 μs nakon dodjeljivanja vrijednosti signalu S1

Primjer 36.

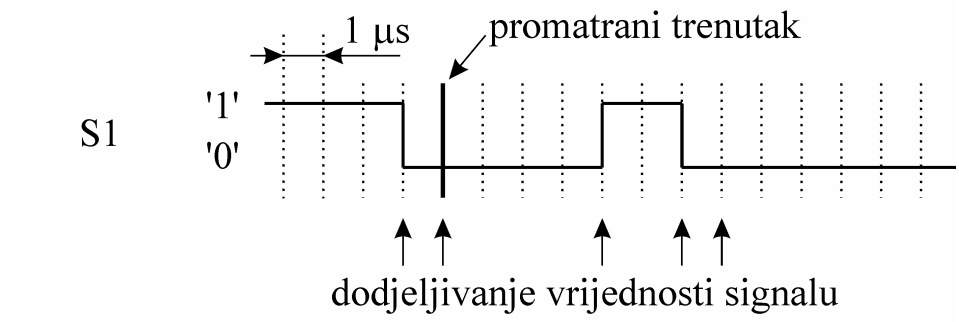
`Signal'transaction`



- **`Signal'transaction`** mijenja stanje nakon svakog dodjeljivanja vrijednosti signalu S1

Primjer 37.

- atributi koji predstavljaju vrijednosti određenih parametara signala

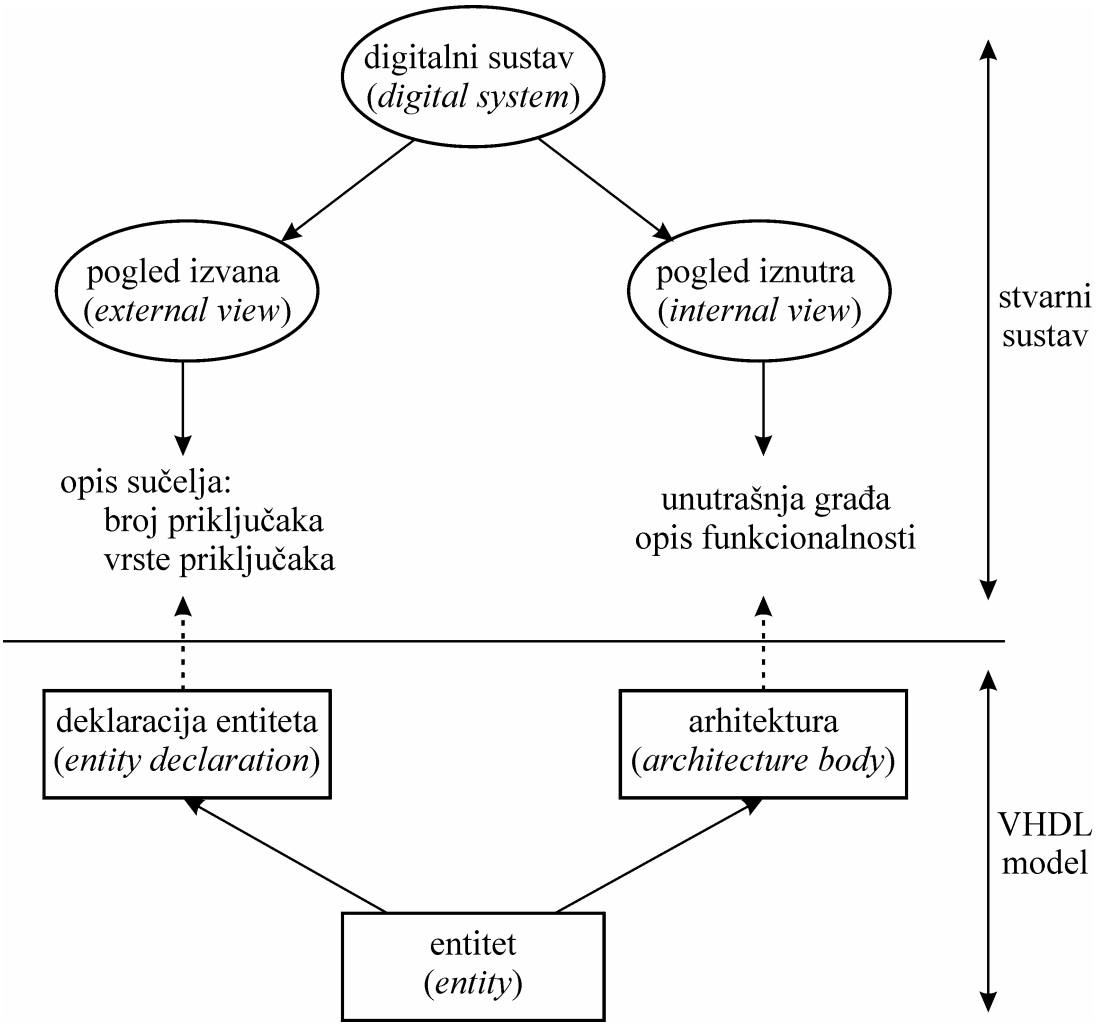



```
-- vrijednosti atributa

      -- Signal'event       = false
      -- Signal'active      = true
      -- Signal'last_event  = 1 us
      -- Signal'last_active = 1 us
      -- Signal'last_value  = '1'
```

Predefinirani atributi signala

Atribut	Rezultat	Tip rezultata	Opis
`Signal'delayed(T)`	implicitni signal	isti tip kao **`Signal`**	signal pomaknut za vrijeme T (kašnjenje za T)
`Signal'stable(T)`	implicitni signal	boolean	**`true`** ako se tokom vremena T nije dogodila PROMJENA vrijednosti signala
`Signal'quiet(T)`	implicitni signal	boolean	**`true`** ako tokom vremena T nije bilo DODJELJIVANJE vrijednosti signalu (<=)
`Signal'transaction`	implicitni signal	bit	alternira vrijednosti (*toggle*) **`'0'`** i **`'1'`** svaki put kad se signalu dodijeli vrijednost
`Signal'event`	skalar	boolean	**`true`** ako se u promatranom trenutku dogodila PROMJENA vrijednosti signala
`Signal'active`	skalar	boolean	**`true`** ako se u promatranom trenutku dogodilo DODJELJIVANJE vrijednosti signalu
`Signal'last_event`	vrijeme	time	vrijeme prije kojeg je došlo do posljednje PROMJENE vrijednosti signala
`Signal'last_active`	vrijeme	time	vrijeme prije kojeg je došlo do posljednjeg DODJELJIVANJA vrijednosti signalu
`Signal'last_value`	skalar	isti tip kao **`Signal`**	vrijednost signala prije njegove posljednje promjene

Građa VHDL modela



Osnovni oblik deklaracija entiteta

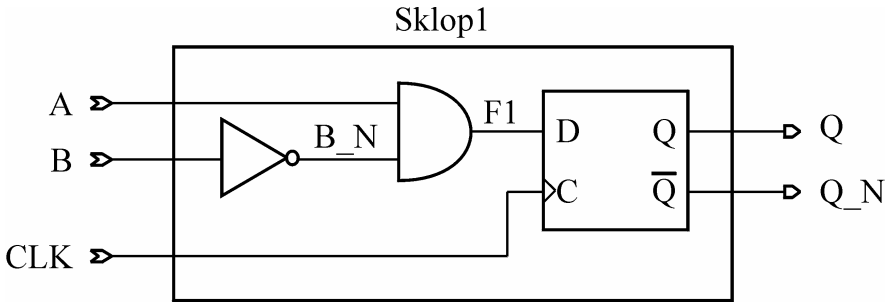
- služi za vanjski opis priključaka
- ne govori ništa o funkcionalnosti entiteta

Osnovni oblik deklaracije entiteta:

```
entity ImeEntiteta is
   port (
      ImePina1: ModPina1 TipPina1:= PocetnoStanje1;
      ImePina2, ImePina3: ModPina23 TipPina23:= PocetnoStanje23;
      -- itd.
      ImePinaN: ModPinaN TipPinaN:= PocetnoStanjeN -- bez ;
        );
end entity ImeEntiteta;
```

- **ModPina** može biti
  in
  out
  inout
  buffer

Primjer 38.




```
-- deklaracija entiteta

entity Sklop1 is
    port ( A, B : in bit;
           CLK  : in bit;
           Q    : out bit;
           Q_N  : out bit);
end entity Sklop1;
```

Arhitektura

- opisuje unutrašnjost sustava
- razlikujemo dva pristupa pisanju arhitektura (vidi uvodna poglavlja)
    - opis ponašanja (*behavioral modeling*)

      ⇒ temeljen na ravnopravnim jednadžbama i slijednom opisu
    - opis veza između podsustava (*structural modeling*)

      ⇒ opis strukture (lista spajanja, *net-list*)

Osnovna građa arhitekture

```
-- Oblik arhitekture koja predstavlja opis ponasanja sustava.

architecture ImeArhitekture of ImeEntiteta is
   -- deklaracije lokalnih tipova i objekata
         -- tipova
         -- konstanti
         -- signala
         -- komponenata
         -- itd.
         -- NE MOGU SE DEKLARIRATI varijable
begin
   -- Opis sustava.
   -- Linije u ovom bloku predstavljaju RAVNOPRAVNE jednadzbe.
   -- Npr.:
         -- dodjeljivanje vrijednosti signalu
         -- proces
         -- blok
         -- opis spajanja komponente
         -- poziv potprograma
         -- zahtjev za provjeru
end architecture ImeArhitekture;
```

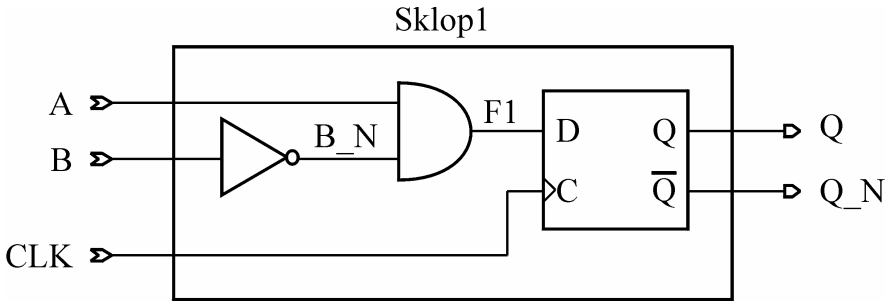
Uočiti

- arhitektura predstavlja niz ravnopravnih jednadžbi (izraza)

  ⇒ IZVRŠAVAJU SE PARALELNO

  ⇒ redoslijed jednadžbi (izraza) u tijelu arhitekture je PROIZVOLJAN
- opis strukture je također građen od ravnopravnih jednadžbi (izraza)

Primjer 39.




```
-- arhitektura koja opisuje ponasanje sklopa

architecture OpisPonasanja of Sklop1 is
   signal B_N, F1: bit;
begin

   -- jednadzbe i proces obraduju se PARALELNO

   B_N <= not B;

   F1  <= A and B_N;

   process (CLK) is -- za proces vidi slijedeca poglavlja
   begin
      if ( CLK='1' ) then
         Q   <= F1;
         Q_N <= not F1;
      end if;
   end process;

end architecture OpisPonasanja;
```

Proces

- služi za opis funkcionalnosti
- proces se izvodi slijedno

Osnovna građa procesa

```
-- Oblik procesa.

labela: process (AktivacijskiSignal1, AS2, ...) is
           -- deklaracije lokalnih tipova i objekata
                -- tipova
                -- konstanti
                -- varijable
                -- itd.
                -- NE MOGU SE DEKLARIRATI signali
        begin
           -- Opis procesa.
           -- Linije u ovom bloku su SLIJEDNI izrazi.
           -- Npr.:
                -- dodjeljivanje vrijednosti signalu
                -- dodjeljivanje vrijednosti varijabli
                -- uvjeti
                -- petlje
                -- odgoda izvrsavanja
                -- poziv potprograma
                -- zahtjev za provjeru
        end process labela;
```

- aktiviranje procesa
  - ako postoji aktivacijska lista (*sensitivity-list*)

    ⇒ krene automatski na početku simulacije

    ⇒ kad dođe do kraja, proces stane i čeka novu aktivaciju (odgađanje (*suspend*))

    ⇒ nova aktivacija je promjena vrijednosti barem jednog aktivacijskog signala
  - ako ne postoji aktivacijska lista

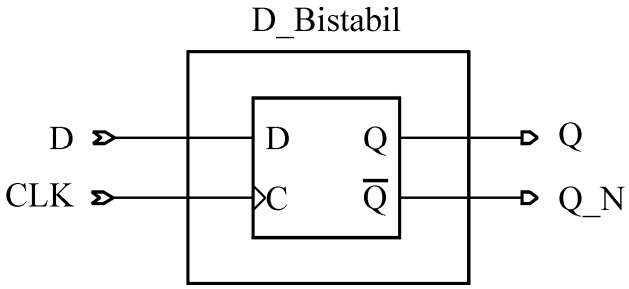
    ⇒ krene automatski na početku simulacije

    ⇒ stane kad naiđe na uvjet za odgodu ,"`wait` uvjet"

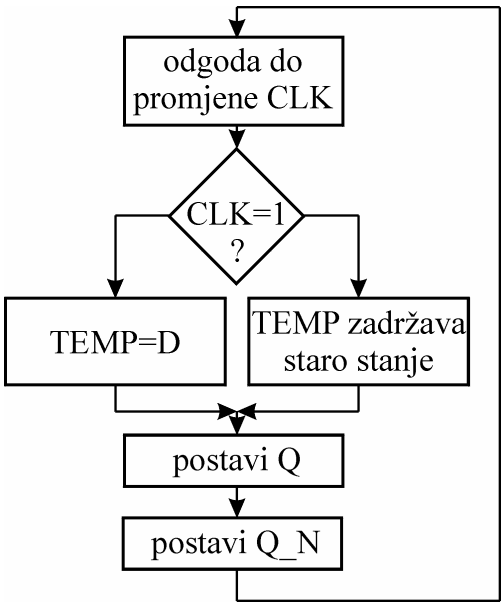
    ⇒ kad se ispuni uvjet, nastavi dalje

    ⇒ kad dođe do kraja, krene iz početka

Primjer 40.

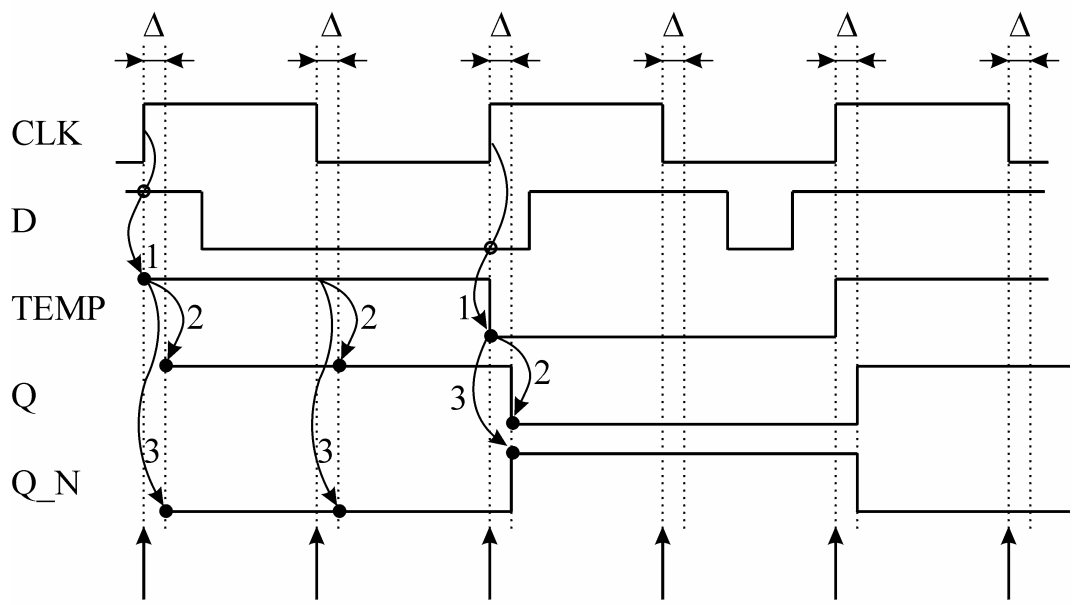



```
-- deklaracija entiteta

entity D_Bistabil is
   port ( D   : in  bit;
          CLK : in  bit;
          Q   : out bit;
          Q_N : out bit);
end entity D_Bistabil;

-- primjer arhitekture koja koristi proces

architecture PonasanjeDB of D_Bistabil is

begin

   process (CLK) is
      -- lokalna varijabla
      variable TEMP: bit;
   begin

      -- slijedni izraz 1
      if CLK='1' then
         TEMP := D;
      end if;

      -- slijedni izraz 2
      Q <= TEMP;

      -- slijedni izraz 3
      Q_N <= not TEMP;

   end process;

end architecture PonasanjeDB;
```



Što će učiniti simulator?

```
process (CLK) is
   -- lokalna varijabla
   variable TEMP: bit;
begin

   -- slijedni izraz 1
   if CLK='1' then
      TEMP := D;
   end if;

   -- slijedni izraz 2
   Q <= TEMP;

   -- slijedni izraz 3
   Q_N <= not TEMP;

end process;
```



trenuci aktivacije procesa

Uočiti

- kad se promijeni bilo koji argument procesa sve jednadžbe se "računaju" u tom trenutku, redom kojim su napisane
- pritom se vrijednosti varijabli mijenjaju u trenutku promjene (:=), a signala nakon vremena Δ (<=)

Napomene

- proces mora imati ili aktivacijsku listu ili barem jedan **`wait`**, ali NE OBOJE
- lokalne varijable zadržavaju staru vrijednost pri ponovnom aktiviranju procesa

Slijedni izrazi

- ispitivanje uvjeta
- odgoda izvršavanja
- petlje

Ispitivanje uvjeta

- **`if` naredba**
- **`case` naredba**

Ispitivanje uvjeta pomoću `if` naredbe

- koristi se za ispitivanje uvjeta
- uvjet može biti složen

Sintaksa `if` naredbe

```
labela: if BooleanIzraz1 then  -- labela nije obavezna
           -- slijedni izrazi

        elsif BooleanIzraz2 then
           -- slijedni izrazi

        elsif BooleanIzraz3 then
           -- slijedni izrazi

        -- itd.

        else
           -- slijedni izrazi

        end if labela;
```

Primjer 41.

Napisati VHDL model dvopoložajnog regulatora s histerezom, koji se koristi za regulaciju temperature. Ulazi u model su željena temperatura, izmjerena temperatura i histereza, a izlaz iz modela je kontrolni signal kojim se uključuje grijanje.

```
-- Dvopolozajni regulator temperature s histerezom

entity RegulatorTemperature is
   port (ZeljenaT, MjerenaT, Histereza: in integer;
         Grijanje: out bit);
end entity RegulatorTemperature;

architecture PrincipRada1 of RegulatorTemperature is
begin
   process (ZeljenaT, MjerenaT, Histereza) is
   begin
      if MjerenaT<=(ZeljenaT-Histereza) then    -- donji prag
         Grijanje <= '1';
      elsif MjerenaT>=(ZeljenaT+Histereza) then -- gornji prag
         Grijanje <= '0';
      end if;
   end process;
end architecture PrincipRada1;
```

Napomene:

- izraz koji se ispituje (**BooleanIzraz**) mora kao rezultat vraćati tip **boolean**
- **elsif** i **else** su opcionalni
- ugnježđivanje **if** naredbi je dozvoljeno, ali se ne preporučuje u fizički ostvarivim modelima
- naredbom **if-elsif** ispituju se redom uvjeti; kad se naiđe na PRVI istinit (**true**), izvrše se odgovarajući slijedni izrazi, te prekine ispitivanje preostalih uvjeta

Ispitivanje uvjeta pomoću `case` naredbe

- koristi se za ispitivanje rezultata jedne operacije
- različiti rezultati uzrokovat će različite akcije

Sintaksa `case` naredbe

```
labela: case Izraz is                    -- labela nije obavezna

           when Izbor1 =>                -- ekvivalentno Izraz=Izbor1
              -- slijedni izrazi

           when Izbor2 =>
              -- slijedni izrazi

           when Izbor3 =>
              -- slijedni izrazi

           -- itd.

           when others =>                -- when others nije obavezno
              -- slijedni izrazi

        end case labela;
```

- znak => označava preusmjeravanje (*redirection*) na željene slijedne izraze
- ako pojedini slučaj izbora "ne radi ništa", tj. ne sadrži slijedne izraze, koristi se slijedni izraz `null`

```
           when Izbor2 =>
              null;
           when Izbor3 =>
              -- slijedni izrazi
```

Primjer 42.

Napisati VHDL model dekodera za pogon 7-segmentnog pokaznika ako se na njemu žele prikazivati dekadske znamenke. Pretpostaviti da su znamenke predstavljene BCD kodom.

```
-- Dekoder BCD u 7-segmentni pokaznik

entity bcd2pokaznik is
   port (BCD      : in  integer range 0 to 9;
         Pokaznik : out integer range 0 to 127);
end entity bcd2pokaznik;


architecture PrincipRada2 of bcd2pokaznik is
begin

   process (BCD) is
   begin
      case BCD is
         when 0 => Pokaznik<=126;     -- 1111110
         when 1 => Pokaznik<=12;      -- 0001100
         when 2 => Pokaznik<=91;      -- 1011011
         when 3 => Pokaznik<=31;      -- 0011111
         when 4 => Pokaznik<=37;      -- 0100101
         when 5 => Pokaznik<=55;      -- 0110111
         when 6 => Pokaznik<=119;     -- 1110111
         when 7 => Pokaznik<=28;      -- 0011100
         when 8 => Pokaznik<=127;     -- 1111111
         when 9 => Pokaznik<=63;      -- 0111111
      end case;
   end process;

end architecture PrincipRada2;
```

Primjer 43.

Napisati VHDL model cjelobrojne aritmetičke jedinice procesora kojom se izvršavaju zbrajanje, oduzimanje, množenje i dijeljene. Pretpostaviti da je enumeracijski tip koji opisuje skup instrukcija procesora deklariran u odgovarajućem paketu.

```
-- cjelobrojna aritmeticka jedinica

-- ovaj tip nalazi se u korisnikovom paketu:
-- type SkupInstrukcija is (LOAD,ADD,SUB,MUL,DIV,NOP,STORE);


entity aritmeticka_jedinica is
   port (Instrukcija : in SkupInstrukcija;
         A, B        : in integer;
         Rezultat    : out integer);
end entity aritmeticka_jedinica;

architecture PrincipRada3 of aritmeticka_jedinica is
begin

   process (Instrukcija) is
   begin
      case Instrukcija is
         when ADD => Rezultat<=A+B;
         when SUB => Rezultat<=A-B;
         when MUL => Rezultat<=A*B;
         when DIV => Rezultat<=A/B;
         when NOP => null;      -- ne radi nista
         when others => null;   -- ne radi nista
      end case;
   end process;

end architecture PrincipRada3;
```

Napomene

- slijedni izrazi mogu se koristiti samo UNUTAR PROCESA
- ako se umjesto **case** koristi **if**, rezultat će funkcijski biti isti, ali se može razlikovati vrijeme potrebno za simulaciju

⇒ prije početka rada s nekim paketom potrebno se je upoznati sa svojstvima tog paketa

Odgoda izvršavanja

- koristi se za odgađanje procesa do ispunjenja zadanog uvjeta
- nakon odgode proces nastavlja s izvršavanjem

Oblici i sintaksa `wait` naredbe

```
-- ceka dok protekne zadano vrijeme
labela: wait for Vrijeme;

-- ceka dok se ne promjeni stanje barem jednog od
-- aktivacijskih signala
labela: wait on AktivacijskiSignali;

-- ceka do trenutka kad dani BooleanIzraz postane true
labela: wait until BooleanIzraz;

-- ceka zauvijek
labela: wait;

-- ceka dok se ne ispuni zadana kombinacija
-- dopustene su sve kombinacije on, until i for
labela: wait on AktivacijskiSignali
             until BooleanIzraz
             for Vrijeme;
```

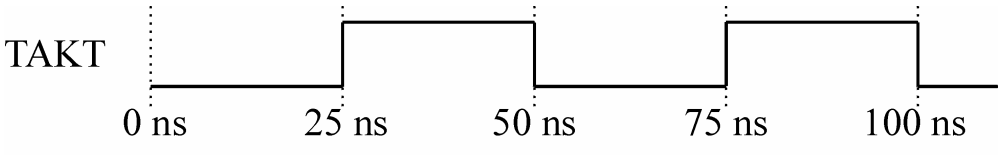
Primjer 44.

```
wait for 10 ns; -- cekaj 10 ns
wait on A, B;   -- cekaj do promjene stanja signala A ili B
wait until RESET='0';   -- cekaj dok RESET postane neaktivan
wait;           -- cekaj zauvijek

-- Cekaj do promjene stanja signala CLK,
-- uz uvjet da u trenutku promjene CLKEN bude jednak 1.
wait on CLK until CLKEN='1';

-- Cekaj do promjene stanja signala Ulaz42.
-- Ako se promjena ne dogodi u narednih 10 ms, idi dalje.
wait on Ulaz42 for 10 ms;

-- Cekaj do promjene stanje signala Ulaz42, uz uvjet
-- da u trenutku promjene ENABLE bude jednak 1.
-- Ako se promjena ne dogodi u narednih 10 ms, idi dalje.
wait on Ulaz42 until ENABLE='1' for 10 ms;
```

Primjer 45.

Pomoću odgovarajućeg procesa modelirati izvor signala takta frekvencije 20 MHz.

```
process is
begin
   TAKT <= '0';
   wait for 25 ns;
   TAKT <= '1';
   wait for 25 ns;
end process;
```



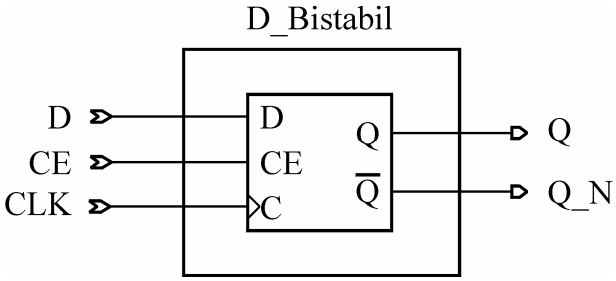
Primjer 46.

```
-- proces bez aktivacijske liste mora imati barem jedan wait

process is
begin
   -- slijedni izrazi
   wait on A, B; -- aktivacijski signali ne idu u zagrade
end process;

-- gornji proces ekvivalentan je slijedecem procesu

process (A, B) is
begin
   -- slijedni izrazi
end process;
```

Primjer 47.

Napisati VHDL model D bistabila opisanog slijedećom tablicom istinitosti

D	CE	CLK	Q_n	$\overline{Q}_n$
X	0	X	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	↑	0	1
1	1	↑	1	0




```
-- Referentni model D_Bistabila s kontrolom upisa

entity D_Bistabil is
    port ( D   : in bit;
           CE  : in bit;    -- Clock Enable
           CLK : in bit;
           Q   : out bit;
           Q_N : out bit );
end entity D_Bistabil;

architecture bistabil_s_CE of D_Bistabil is
begin

   process is
   begin
      Q   <= D;
      Q_N <= not D;
      wait on CLK until (CLK='1' and CE='1');
   end process;

end architecture bistabil_s_CE;
```

Napomene:

- `wait for` se koristi u generiranju ispitnog okruženja
- `wait for` se ne koristi u opisu sklopovlja

  ⇒ ova naredba vremenski suspendira proces, a ne "gasi" sklop

Petlje

- tipovi petlji
  - *loop* - beskonačna petlja
  - *for loop* - petlja s određenim brojem prolaza
  - *while loop* - petlja s ispitivanjem uvjeta

Loop - beskonačna petlja

Osnovni oblik beskonačne petlje

```
labela: loop
           -- slijedni izrazi
        end loop labela;
```

Primjer 48.

Napisati VHDL model generatora adresa za prozivnu tabelu (*look-up-table*). Generator na svom izlazu daje 16-bitnu adresu koja ponovo kreće od 0 nakon što dođe do posljednje adrese. Povećanje adrese izvesti na rastući brid signala takta.

```
entity GeneratorAdresa is
   port( CLK    : in bit;
         -- 16 bitna ROM adresa
         Adresa : out integer range 0 to 65535
       );
end entity GeneratorAdresa;

architecture GA_ReferentniModel of GeneratorAdresa is

begin

   process is
      variable Brojac: integer range 0 to 65535:=0;
   begin
      loop
         wait on CLK until (CLK='1'); -- cekaj rastuci brid CLK
         Adresa<=Brojac;
         Brojac:=Brojac+1;  -- novo stanje programskog brojila
                            -- nakon 65535 slijedi 0
      end loop;
   end process;

end architecture GA_ReferentniModel;
```

For loop - petlja s određenim brojem prolaza

Osnovni oblik petlje

```
labela: for Parametar in Interval loop
           -- slijedni izrazi
        end loop labela;
```

Primjer 49.

```
for I in 0 to 15 loop      -- I poprima vrijednosti od 0 do 15
   -- slijedni izrazi
end loop;

for I in 15 downto 0 loop -- I poprima vrijednosti od 15 do 0
   -- slijedni izrazi
end loop;

for I in integer loop      -- I poprima vrijednosti
                           -- od integer'low do integer'high
   -- slijedni izrazi
end loop;
```

Primjer 50.

```
type Dani is (PON, UTO, SRI, CET, PET, SUB, NED);
...
for D in Dani'PON to Dani'PET loop -- D poprima vrijednosti
                                   -- od PON do PET
   -- slijedni izrazi
end loop;

for D in Dani loop -- D poprima vrijednosti od PON do NED
   -- slijedni izrazi
end loop;
```

Uočiti

- **`Parametar`** ne treba eksplicitno deklarirati
- tip parametra određen je DISKRETNIM intervalom koji slijedi nakon **`in`**
- parametar se ne može mijenjati unutar petlje (ponaša se kao konstanta)

While loop - petlja s određenim brojem prolaza

Osnovni oblik petlje

```
labela: while Uvjet loop
           -- slijedni izrazi
        end loop labela;
```

- petlja se izvršava dok je **`Uvjet`** ispunjen
- **`Uvjet`** je izraz koji daje tip **`boolean`**
- unutar petlje može se mijenjati neki parametar koji sačinjava uvjet

Primjer 51.

```
while Brojac<20 loop      -- kad se napuni gajba, izadi iz petlje

   Brojac:=Brojac+PristigleBocePiva;

   -- slijedni izrazi

end loop;
```

Exit - izlaz iz petlje

Sintaksa `exit` naredbe

```
-- bezuvjetni izlaz iz petlje

exit;
exit labela;

-- uvjetni izlaz iz petlje

exit when Uvjet;
exit labela when Uvjet;
```

- **`Uvjet`** je izraz koji daje tip **`boolean`**

Primjer 52.

```
-- Ilustracija izlaska iz jedne ili vise petlji
-- Ilustracija upotrebe labela

petlja1: loop
   ...
   exit;  -- izadi iz petlje
   ...
   petlja2: loop
      ...
      exit;  -- izadi samo iz unutrasnje petlje
      ...
      exit when C>0;  -- ako je C>0, izadi iz unutrasnje petlje
      ...
      exit petlja1;   -- izadi iz vanjske i unutrasnje petlje
      ...
      exit petlja1 when C>0; -- ako je C>0, izadi iz obje petlje
      ...
   end loop petlja2;
   ...
end loop petlja1;

nastavak: A<=B;
...
-- itd.
```

Napomena

- **exit** se može primijeniti na sve oblike petlji (**loop**, **for loop** i **while loop**)

Uočiti

- slijedeći izrazi su ekvivalentni

```
exit when Uvjet;

if Uvjet
   exit;
end if;
```

- izlaziti se može iz jedne ili više petlji istovremeno

Next - prekid tekućeg prolaza petlje

- **next** prekida tekući prolaz petlje i započinje novi prolaz

Sintaksa **next** naredbe

```
-- bezuvjetni prekid prolaza

next;
next labela;

-- uvjetni prekid prolaza

next when Uvjet;
next labela when Uvjet;
```

- **Uvjet** je izraz koji daje tip **boolean**

Primjer 53.

```
-- Ilustracija prekida tekuceg prolaza petlje

petlja: loop
   ...
   next;        -- kreni na novi prolaz petlje
   ...
   next when C>0;    -- ako je C>0, kreni na novi prolaz petlje
   ...
   next petlja when C>0; -- isto sto i prethodni next
   ...
end loop petlja;
```

Napomena

- treba IZBJEGAVATI prekid prolaza više petlji istovremeno (nepregledno je !!!)

Uočiti

- slijedeći izrazi su ekvivalentni

```
next when Uvjet;

if Uvjet
   next;
end if;
```

- `next` i `exit` imaju "jednaku" sintaksu

Kašnjenje

- Δ kašnjenje
- kašnjenje tipa **inertial**
- kašnjenje tipa **transport**

Kašnjenje tipa *inertial*

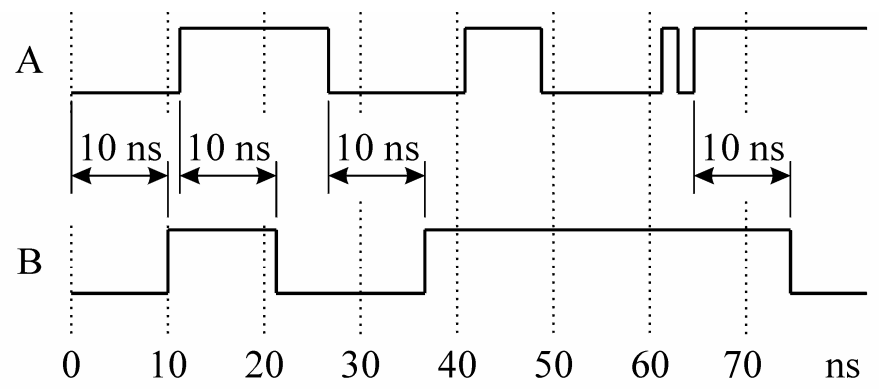
- služi za modeliranje kašnjenja digitalnih sklopova

Sintaksa kašnjenja tipa *inertial*

```
-- izlazni signal jednak je ulaznom zakasnjenom za Vrijeme
-- izlazni signal "potisnut" je ako je impuls kraci od Vrijeme
IzlazniSig <= inertial UlazniSig after Vrijeme;

-- rijec inertial moze se izostaviti
IzlazniSig <= UlazniSig after Vrijeme;

-- izlazni signal jednak je ulaznom zakasnjenom za Vrijeme
-- izlazni signal "potisnut" je ako je impuls kraci od Vrijeme1
IzlazniSig <= reject Vrijeme1 inertial UlazniSig after Vrijeme;
```

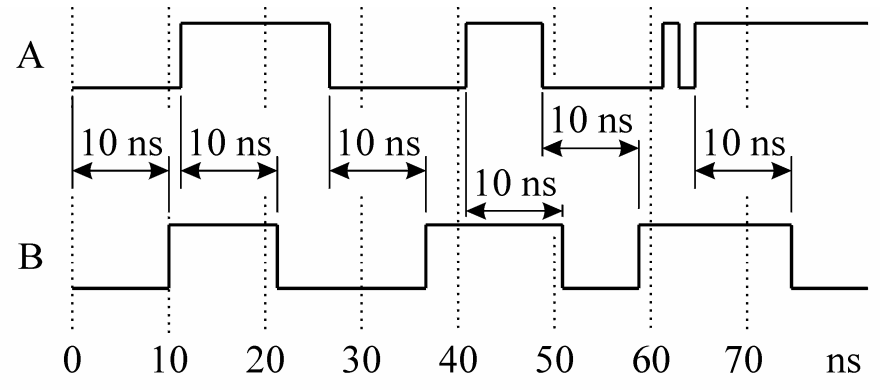
Primjer 54.

```
-- pretpostavka da su A i B signali tipa bit

process (A) is
begin
   B <= inertial not A after 10 ns;
end process;
```



Primjer 55.

```
-- pretpostavka da su A i B signali tipa bit

process (A) is
begin
   B <= reject 5 ns inertial not A after 10 ns;
end process;
```



Uočiti

- kašnjenje tipa inertial ne "propušta" prekratke impulse

  ⇒ događaji moraju biti razmaknuti u vremenu više nego što je specificirano vrijeme
- slijedeći izrazi su ekvivalentni

```
B <= A after 10 ns;
B <= inertial A after 10 ns;
B <= reject 10 ns inertial A after 10 ns;
```

- kašnjenje za Δ se podrazumijeva

```
-- B <= A   znaci   B <= A after delta
```

- stanje signala B u trenutku t=0 određeno je inicijalnim stanjem (*default*) enumeracijskog tipa (u ovom slučaju je to bit)

Kašnjenje tipa *transport*

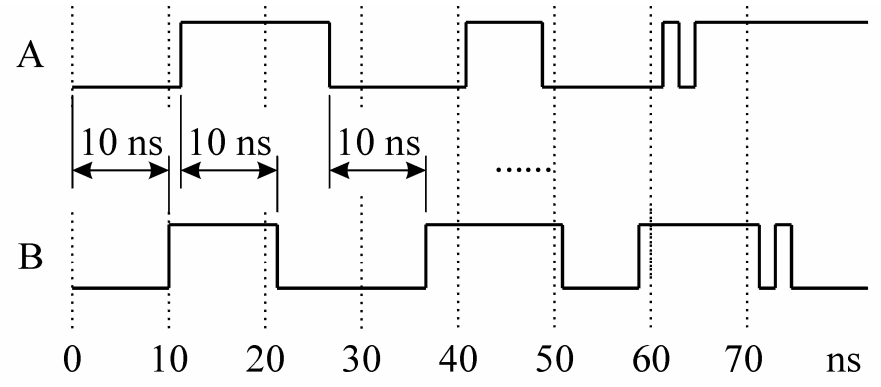
- "obično" kašnjenje; izlaz kasni za ulazom određeno vrijeme

Sintaksa kašnjenja *transport*

```
-- izlazni signal jednak je ulaznom zakasnjenom za Vrijeme
-- nema potiskivanja bez obzira koliko su kratki impulsi

IzlazniSignal <= transport UlazniSignal after Vrijeme;
```

Primjer 56.

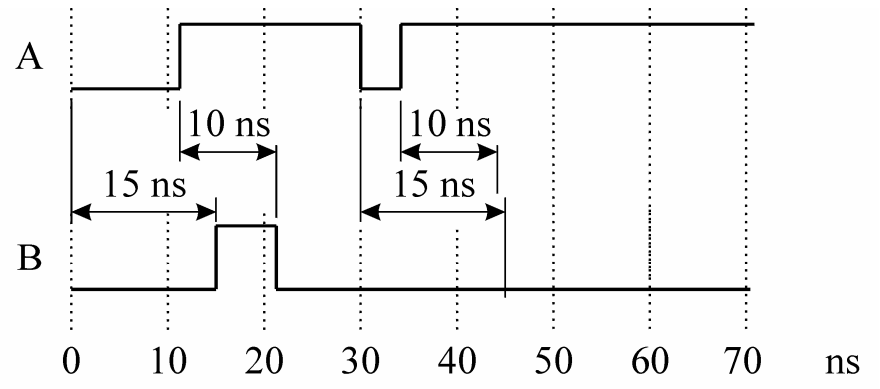
```
-- pretpostavka da su A i B signali tipa bit

process (A) is
begin
   B <= transport not A after 10 ns;
end process;
```



- obrada međusobno proturječnih zahtjeva

Primjer 57.

```
-- tHL razlicito od tLH

-- pretpostavka da su A i B signali tipa bit

process (A) is
begin
   if A='1' then
      B <= transport not A after 10 ns;   -- tHL
   else
      B <= transport not A after 15 ns;   -- tLH
   end if;
end process;
```



Zahtjevi za provjeru

- služe za provjeru ispravnosti dizajna
  - funkcijska provjera
  - provjera vremenskih odnosa

Sintaksa zahtjeva za provjeru

```
labela: assert BooleanIzraz
          report "Ovo zelimo ispisati na ekranu simulatora"
          severity TezinaGreske;  -- note,warning,error,failure
```

- ako je **`BooleanIzraz=false`**, ispisuje se poruka i greška opisuje odgovarajućom težinom (vidi tip **`severity_level`**)
- ako je izostavljen **`report`**, ispisuje se **`"Assertion violation."`**
- ako je izostavljen **`severity`**, uzima se **`severity error`**

- simulator se može podesiti da prekine simulaciju ovisno o postavljenoj vrijednosti **`TezinaGreske`**

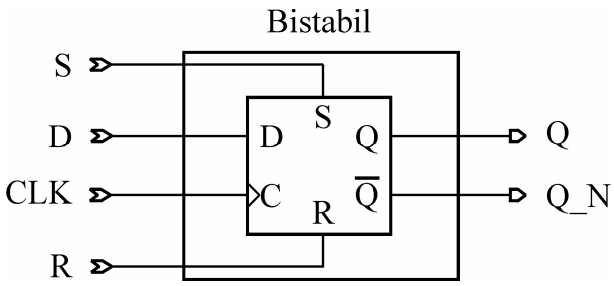
Zahtjev za provjeru može biti izveden kao

- dio procesa, tj. kao slijedni izraz
- jedna od ravnopravnih jednadžbi

Primjer 58.

Napisati VHDL model D bistabila opisanog slijedećom tablicom istinitosti

S	R	D	CLK	Q_n	$\overline{Q}_n$
0	0	0	↑	0	1
0	0	1	↑	1	0
0	1	X	↑	0	1
1	0	X	↑	1	0
1*	1*	X	↑		



* zabranjena kombinacija

```
entity Bistabil is
   port ( CLK, D, R, S: in bit;
          Q, Q_N: out bit );
end entity Bistabil;

architecture OpisRada of Bistabil is
begin
   process is
   begin

      assert (not(R='1' and S='1'))   -- assert kao dio procesa
         report "Zabranjena kombinacija na bistabilu!"
         severity ERROR;

      if R='1' then          -- "reset" bistabila
         Q   <= '0';
         Q_N <= '1';
      elsif S='1' then       -- "set" bistabila
         Q   <= '1';
         Q_N <= '0';
      else                   -- upis u bistabil
         Q   <= D;
         Q_N <= not D;
      end if;

      wait until CLK='1';
      -- isto sto i wait on CLK until CLK='1';

   end process;

end architecture OpisRada;
```

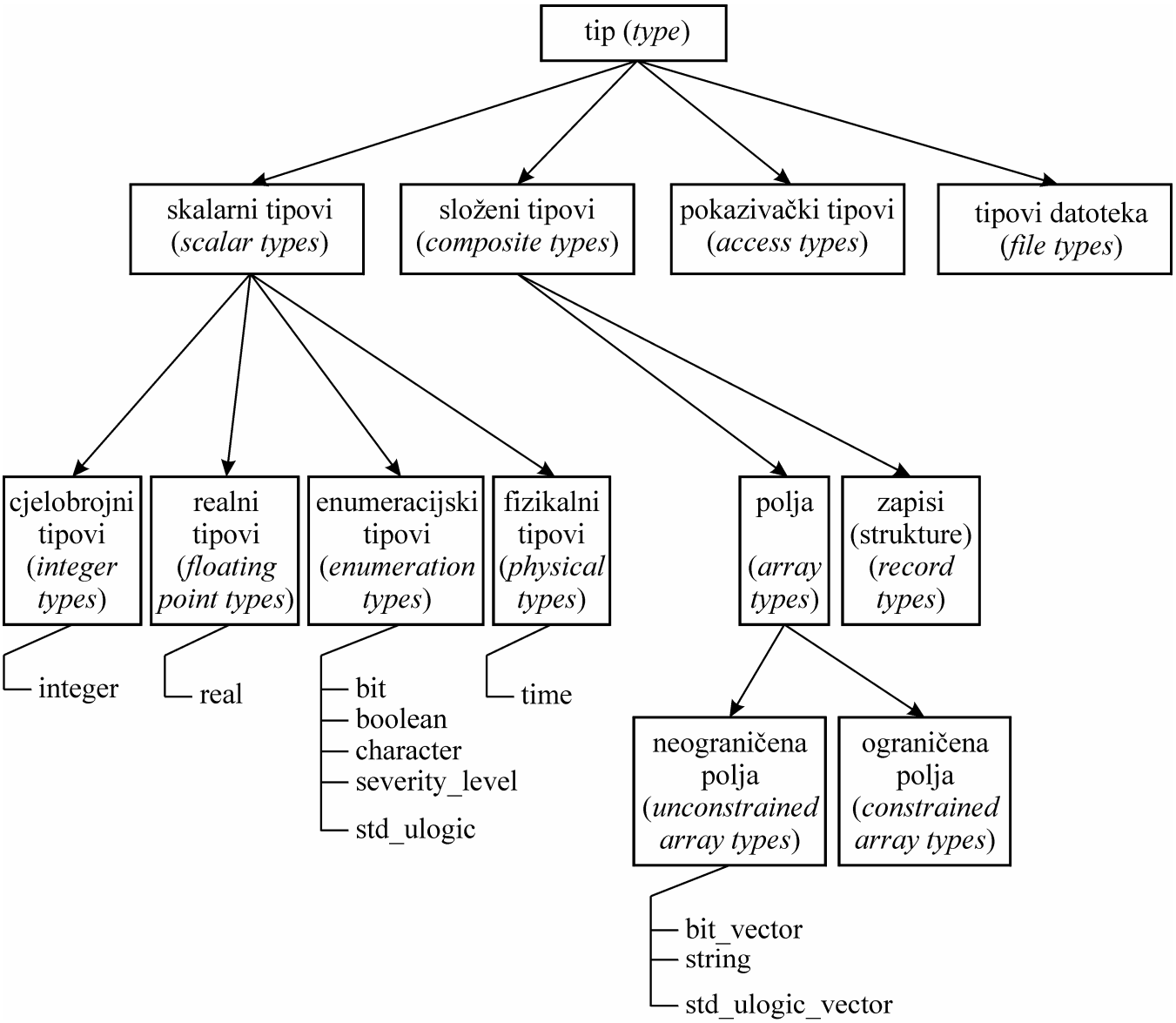
Primjer 59.

U VHDL model dekodera za pogon 7-segmentnog pokaznika ugraditi provjeru pojave nule.

```
entity bcd2pokaznik is
   port (BCD      : in  integer range 0 to 9;
         Pokaznik : out integer range 0 to 127);
end entity bcd2pokaznik;

architecture PrimjerBCD of bcd2pokaznik is
begin

   -- assert kao ravnopravan izraz
   assert (not (BCD=0))
      report "Pojavio se BCD broj jednak nuli!"
      severity NOTE;

   process (BCD) is
   begin
      case BCD is
         when 0 => Pokaznik<=126;     -- 1111110
         when 1 => Pokaznik<=12;      -- 0001100
         when 2 => Pokaznik<=91;      -- 1011011
         when 3 => Pokaznik<=31;      -- 0011111
         when 4 => Pokaznik<=37;      -- 0100101
         when 5 => Pokaznik<=55;      -- 0110111
         when 6 => Pokaznik<=119;     -- 1110111
         when 7 => Pokaznik<=28;      -- 0011100
         when 8 => Pokaznik<=127;     -- 1111111
         when 9 => Pokaznik<=63;      -- 0111111
      end case;
   end process;

end architecture PrimjerBCD;
```

Složeni tipovi objekata

Pregled tipova



- polje - skup objekata istog tipa

  ⇒ npr. polje integera, polje bitova, i sl.

- zapis - skup objekata različitog tipa

  ⇒ npr. vrijeme, integer, bit

Polja

- prema dimenziji
    - jednodimenzionalna polja
    - višedimenzionalna polja
- prema veličini
    - neograničena polja
    - ograničena polja

Deklaracija tipa koji predstavlja OGRANIČENO POLJE:

```
type ImeTipa is
    array (TipIndeksa range DoGr to GoGr) of TipElementa;

type ImeTipa is
    array (TipIndeksa range GoGr downto DoGr) of TipElementa;
```

- **DoGr** i **GoGr** predstavljaju donju i gornju graničnu vrijednost indeksa
- **TipIndeksa** mora odgovarati DISKRETNOM SKALARNOM TIPU

Primjer 60.

```
-- deklaracija tipa polja, ako je indeks tipa INTEGER

type Sabirnica8 is
     array (0 to 7) of bit;        -- rastuci indeksi polja

type Sabirnica15d is
     array (15 downto 0) of bit;   -- padajuci indeksi polja

type Vektor is
     array (10 downto 1) of positive;
   -- vidi paket STANDARD u kojem se nalazi:
      -- subtype natural is integer range 0 to integer'high;
      -- subtype positive is integer range 1 to integer'high;

type CjelobrojnoPolje is
     array (0 to 15) of integer range 0 to 255;
```

Primjer 61.

```
-- dodjeljivanje vrijednosti POLJU BITOVA

type Sabirnica8 is
     array (0 to 7) of bit;       -- rastuci interval

variable InternaKontrola: Sabirnica8;

InternaKontrola(0) := '1';
InternaKontrola := "0101_0101"; -- _ se ignorira
InternaKontrola := ('0','1','0','1','0','1','0','1');
InternaKontrola := (3=>'0', 6=>'1', others=>'0');
```

- podaci se mogu unositi

```
B"0101_0101"    -- binarno
O"77"           -- oktalno
X"8F"           -- heksadecimalno
```

Primjer 62.

```
-- dodjeljivanje vrijednosti CJELOBROJNOM POLJU

type CjelobrojnoPolje is
     array (0 to 15) of integer range 0 to 255;

variable Memorija: CjelobrojnoPolje;

Memorija(6) := 33;
Memorija := (42,42,42,42,42,42,42,42,42,42,42,42,42,42,42,42);
```

- slično se radi i za ostale tipove polja

Primjer 63.

```
-- deklaracija tipa polja, ako je indeks ENUMERACIJSKOG tipa

type Ulazi is (UL0,UL1,UL2,UL3,UL4);
type BrojacImpulsa is
     array (Ulazi range UL2 to UL4) of natural;

variable BrojImpulsa: BrojacImpulsa;

BrojImpulsa(UL4) := 0;
BrojImpulsa(UL2) := BrojacImpulsa(UL2)+1;
```

Primjer 64.

Napisati VHDL model ROM memorije koja sadrži 16 riječi širine 8 bitova. Pretpostaviti da su izlazna pojačala na podatkovnim linijama uvijek otvorena.

```
entity ReadOnlyMemory is
    port ( ADR  : in  integer range 0 to 15;
           DATA : out integer range 0 to 255 );
end entity ReadOnlyMemory;

architecture ROM16x8 of ReadOnlyMemory is
begin

   process (ADR) is
      type ROM is array (0 to 15) of integer range 0 to 255;
      variable Program: ROM := (33,120,200,2,241,87,34,211,
                                5,26,123,34,76,146,203,189);
   begin
      DATA <=  Program(ADR);
   end process;

end architecture ROM16x8;
```

- slučaj koji obuhvaća stanje visoke impedancije podatkovnih linija izvodi se pomoću priključaka tipa `std_logic` (vidi daljnji tekst)

Deklaracija tipa koji predstavlja NEOGRANIČENO POLJE:

```
type ImeTipa is
     array (TipIndeksa range <>) of TipElementa;
```

- **TipIndeksa** mora odgovarati DISKRETNOM SKALARNOM TIPU
- oznaka **<>** označava da će **range** biti definiran kasnije (npr. u deklaraciji varijable)

Primjer 65.

```
-- deklaracija tipova koji predstavljaju neogranicena polja

type BlokBitova is
     array (natural range <>) of bit;

type BlokBajtova is
     array (natural range <>) of integer range 0 to 255;

-- deklaracija varijabli

variable UlazniRegistar: BlokBitova(0 to 7);
   -- varijabla UlazniRegistar je polje od 8 bitova:
   -- UlazniRegistar(0),...,UlazniRegistar(7)
variable Stog: BlokBajtova(15 downto 0);
   -- varijabla Stog je polje od 16 bajtova:
   -- Stog(15),..., Stog(0)
```

Predefinirani tipovi polja

```
-- ovaj tip deklariran je u bilioteci STD, paketu STANDARD

type string is array (positive range <>) of character;

type bit_vector is array (natural range <>) of bit;

-- ovaj tip deklariran je u bilioteci IEEE,
-- paketu std_logic_1164

type std_ulogic_vector is
     array (natural range <>) of std_ulogic;
```

Uočiti

- polja su opisana pomoću
    - indeksa
    - elemenata
- indeksi mogu biti BILO KOJEG SKALARNOG TIPA
- elementi mogu biti BILO KOJEG TIPA

Atributi polja

- Predefinirani atributi jednodimenzionalnih polja

Atribut	Opis
`Tip'left`	lijeva granica vrijednosti indeksa
`Tip'right`	desna granica vrijednosti indeksa
`Tip'low`	najmanja vrijednost indeksa
`Tip'high`	najveća vrijednost indeksa
`Tip'range`	interval indeksa
`Tip'reverse_range`	reverzirani interval indeksa
`Tip'length`	broj elemenata indeksa
`Tip'ascending`	`true` za rastući interval (`to`)

- atributi mogu biti primijenjeni ne samo na tip nego i na objekt

Primjer 66.

```
type Blok is array (7 downto 0) of integer range 0 to 255;

variable Stog: Blok;

-- vrijednosti atributa za varijablu Stog
      -- Stog'left=7
      -- Stog'right=0
      -- Stog'low=0
      -- Stog'high=7
      -- Stog'range=7 downto 0
      -- Stog'reverse_range= 0 to 7
      -- Stog'length=8
      -- Stog'ascending=false
```

Rad s komponentama

- služi za spajanje prethodno definiranih komponenata
- "ugradnja" komponente u model sastoji se od
    - deklaracije komponente (*component declaration*)
    - liste spajanja komponente (*component instantiation*)

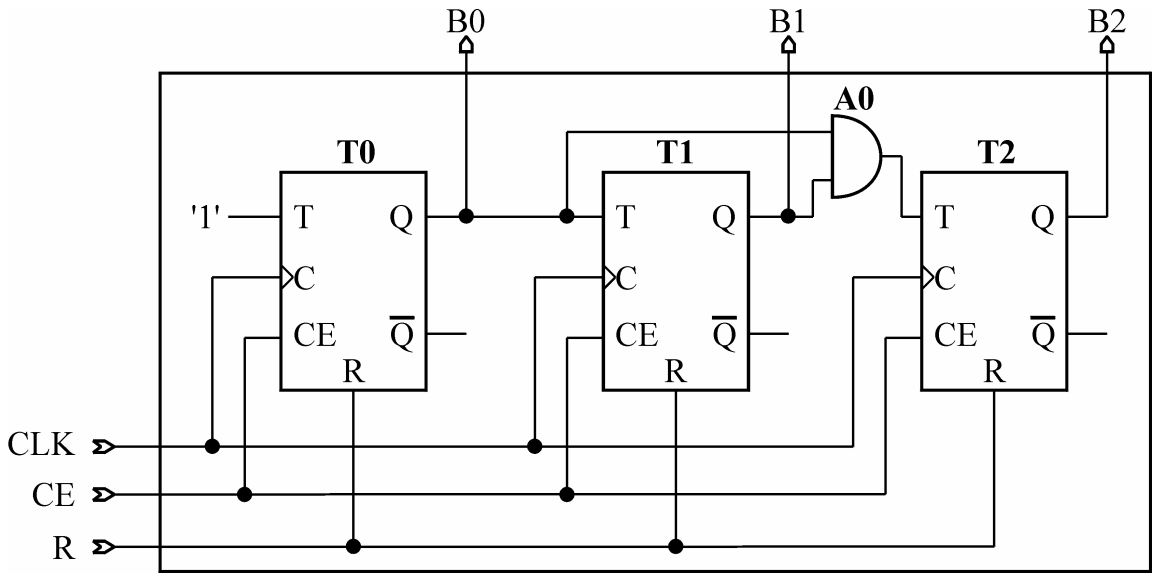
Osnovna sintaksa spajanja komponente u model

```
architecture SintaksaSpojaKomponente of X is
    .
    .
    -- DEKLARACIJA KOMPONENTE
    -- Ako se deklaracija nalazi u paketu koji je ukljucen
    -- kroz neku od biblioteka, ova deklaracija se izostavlja.
    component ImeKomponente
       port(OpisPrikljucaka);
    end component;
    .
    .
begin
    .
    .
    -- SPAJANJE KOMPONENTE
    labela: component ImeKomponente port map (ListaSpajanja);
    .
    .
end architecture SintaksaSpojaKomponente;
```

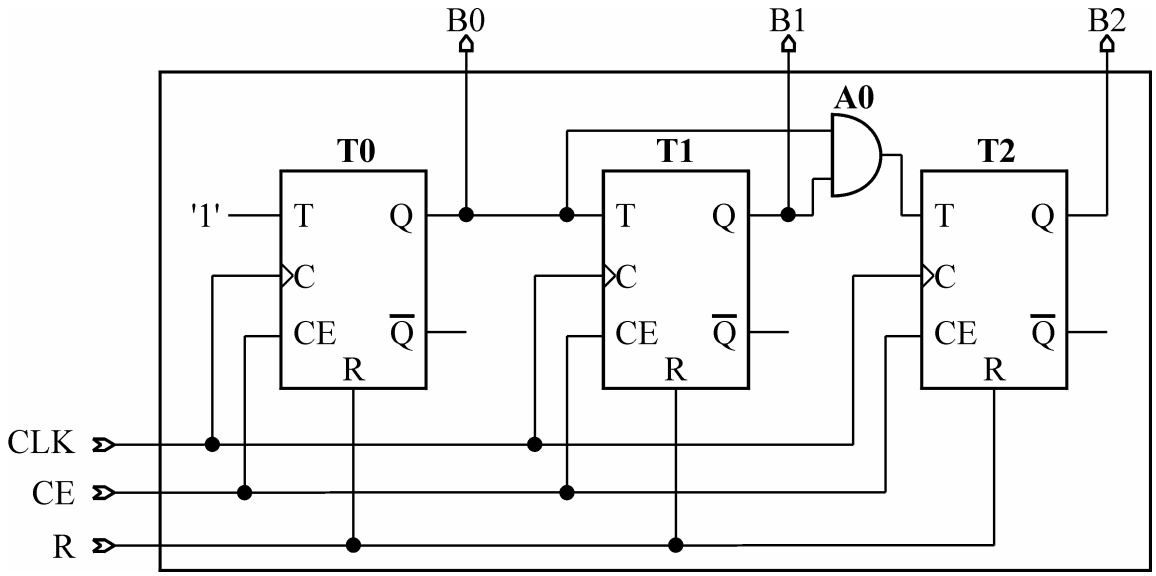
Primjer 67.

Napisati VHDL model 3-bitnog sinkronog brojila s kontrolom upisa i sinkronim brisanjem. Koristiti strukturni stil, uz korištenje postojećeg T-bistabila i dvoulaznih I vrata. Dani su shema sklopa i tablica istine T-bistabila.

T	R	CE	CLK	Q_n	$\overline{Q}_n$
X	1	X	↑	0	1
X	0	0	X	Q_{n-1}	$\overline{Q}_{n-1}$
0	0	1	↑	Q_{n-1}	$\overline{Q}_{n-1}$
1	0	1	↑	$\overline{Q}_{n-1}$	Q_{n-1}




```
-- deklaracija brojila

entity Sinkrono_brojilo is
   port ( CE   : in  bit;
          CLK  : in  bit;
          R    : in  bit;
          B    : out bit_vector(2 downto 0) ); -- B(2) je MSB
end entity Sinkrono_brojilo;
```




```
-- arhitektura brojila

architecture Brojilo_3b of Sinkrono_brojilo is

   component T_Bistabil        -- radi na rastuci brid CLK
      port ( T   : in  bit;
             CE  : in  bit;    -- CLK Enable
             C   : in  bit;
             R   : in  bit;
             Q   : out bit;
             Q_N : out bit );
   end component;

   component AND2
      port ( A, B : in  bit;
             Z    : out bit );
   end component;

   signal S1: bit;
   signal Q : bit_vector(2 downto 0);  -- Q(2) je MSB

begin

   A0: component AND2 port map (Q(0),Q(1),S1);
          -- labele su ovdje obavezne
   T0: component T_Bistabil port map ('1',CE,CLK,R,Q(0),open);
          -- open znaci da prikljucak nije spojen
   T1: component T_Bistabil port map (Q(0),CE,CLK,R,Q(1),open);
   T2: component T_Bistabil port map (S1,CE,CLK,R,Q(2),open);

   B <= Q;

end architecture Brojilo_3b;
```