Uvod u mikroračunarsku elektroniku

Predavanje II

Sadržaj predavanja

- Leksički elementi
- Konstante, promenljive, signali
- Skalarni tipovi
- Atributi skalarnih tipova
- Izrazi i operatori

Leksički elementi i sintaksa

Leksički elementi

- Na narednim slajdovima obradićemo osnovne leksičke elemente VHDL jezika:
  - komentare,
  - identifikatore,
  - rezervisane reči,
  - specijalne simbole,
  - brojeve,
  - karaktere,
  - stringove i
  - bit stringove.

Komentari

- VHDL model sastoji se od određenog broja linija teksta
- Komentar može biti pridodat liniji pišući dve povlake zajedno, koje su zatim praćene tekstom komentara. Na primer:

```
... linija VHDL zapisa ... -- komentar
```

- Komentar se proteže od dve povlake do kraja reda (linije) i može sadržati bilo koji tekst, obzirom da formalno nije deo VHDL modela
- Kod modela može sadržati prazne linije i linije koje sadrže samo komentare, počinjući sa dve povlake
- Možemo pisati dugačke komentare koji se protežu na više linija, pri čemu svaka od njih mora početi sa dve povlake, na primer:

```
-- Naredni kod modeluje
-- upravljački deo sistema
... neki VHDL kod ...
```

Identifikatori I

- Identifikatori se koriste da imenuju objekte u VHDL modelu

- Dobra praksa je da se koriste imena koja upućuju na svrhu objekta, pa stoga VHDL dozvoljava imena proizvoljne dužine

- Međutim, postoje određena pravila u vezi sa formiranjem identifikatora. Običan identifikator:
  - može sadržati samo slova alfabeta ('A' do 'Z' i 'a' do 'z'), decimalne cifre ('0' do '9') i karakter '_';
  - mora početi sa slovom alfabeta;
  - ne sme završiti sa karakterom '_'; i
  - ne sme sadržati dva uzastopna karaktera '_'.

- Nekoliko primera validnih identifikatora:

  A X0 counter Next_Value generate_read_cycle

Identifikatori II

- Primeri napravilnih identifikatora bili bi:
  - last@value -- sadrži nedozvoljeni karakter za identifikator
  - 5bit_counter -- počinje sa nealfabetnim karakterom
  - A0 -- počinje sa karakterom '_'
  - A0_ -- završava se sa karakterom '_'
  - clock__pulse -- dva uzastopna karaktera '_'

- Treba napomenuti da se mala i velika slova ne razlikuju, tako da su identifikatori *cat* i *Cat* identični

- Karakteri '_' su bitni tako da su identifikatori *This_Name* i *ThisName* različiti

Rezervisane reči

- Neki identifikatori, koji se zovu *rezervisane reči* ili *ključne reči*, rezervisani su za posebnu namenu u VHDL
- Oni se koriste da označe specifične konstrukcije koje formiraju model, tako da ih ne možemo koristiti kao identifikatore za objekte koje mi definišemo

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

Specijalni simboli

- VHDL koristi određeni broj specijalnih simbola za označavanje operatora, da razdvaji delove jezičkih konstrukcija i za punktuaciju

- Neki od specijalnih simbola sastoje se samo iz jednog karaktera. To su:

& ' ( ) * + , - . / : ; < > = |

- Drugi specijalni simboli sastoje se iz parova karaktera

- Dva karaktera moraju se nalaziti neposredno jedan do drugog. Ovi simboli su:

=> ** := /= >= <= <>

Brojevi I

- Postoje dve forme brojeva koje se mogu koristiti u VHDL kodu: *celobrojni literali* i *realni literali*
- Celobrojni literal jednostavno predstavlja celi broj i sastoji se od cifara bez decimalne tačke
- Realni literali, sa druge strane, mogu predstavljati *razlomljene* brojeve
- Oni uvek sadrže decimalnu tačku, kojoj prethodi barem jedna cifra i praćena je barem jednom cifrom
- Realni literali predstavljaju aproksimaciju realnih brojeva

Brojevi II

- Primeri decimalnih celobrojnih literala su

23 0 146

- Treba primetiti da –10, na primer, nije celobrojni literal
- On predstavlja kombinaciju operacije negacije i celobrojnog literala 10
- Primeri realnih literala su

23.1 0.0 3.14159

Brojevi III

- I celobrojni i realni literali mogu koristiti eksponencijalnu motaciju, u kojoj nakon broja dolazi slovo 'E' ili 'e', i vrednost eksponenta
- Ovo označava stepen broja 10 sa kojim treba pomnožiti broj
- Za celobrojne literale, eksponent ne sme biti negativan, dok kod relanih literala, on može biti pozitivan i negativan
- Primeri celobrojnih literala koji koriste eksponencijalnu notaciju su

  46E5 1E+12 19e00

- Primeri realnih literala koji koriste eksponencijalnu notaciju su

  1.234E09 98.6E+21 34.0e-08

Brojevi IV

- Ceobrojni i realni literali mogu biti izraženi u osnovi različitoj od 10
- U stvari, osnova može biti bilo koji ceo broj između 2 i 16
- Da bi smo to uradili, pišemo broj okružen sa karakterima '#', kojem prethodi osnova
- Za osnove veće od 9 koriste se slova 'A' do 'F' (ili 'a' do 'f') da predstave brojeve od 10 do 16
- Na primer, evo nekoliko načina na koje je moguće zapisati vrednost 253:

  2#11111101# 16#FD# 16#fd# 8#0375#

- Slično, vrednost 0.5 može se predstaviti kao:

  2#0.100# 8#0.4# 12#0.6#

Brojevi V

- Na kraju, da bi se poboljšala čitljivost dugačkih brojeva, možemo uključiti karakter '_' kao separator između cifara

- Pravila za njihovo korišćenje slična su onim koja važe kod identifikatora; ne smeju se pojaviti na početku i kraju broja, ne mogu biti dva uzastopna karaktera

- Na primer:

123_456 3.141_592_6 2#1111_1100_0000_0000#

Karakteri

- Karakter može biti zapisan u VHDL na taj način što se obuhvati sa jednostrukim znacima navoda

- Bilo koji karakter iz standardnog karakter skupa (uključujući i ' ' karakter) može se zapisati na ovaj način

- Neki primeri su:
  - 'A' -- veliko slovo
  - 'z' -- malo slovo
  - ',' -- karakter za punktuaciju, zarez
  - '"' -- karakter za punktuaciju, jednostruki znak navoda
  - ' ' -- separatorski karakter space

Stringovi I

- String literal predstavlja sekvencu karaktera obuhvacenu sa dvostrukim znacima navoda

- String može imati proizvoljan broj karaktera (uključujući i nulu), ali ceo mora stati u jednu liniju. Primeri su:

  "String”

  "Možemo uključiti svaki proizvoljan karakter (npr. % $ & *) u string!!"

  "00001111ZZZZ"

  "" -- prazan string

- Ako je potrebno da u string uključimo dvostruke navode, pišemo dva znaka dvostrukog navoda zajedno. Na primer:

  "String unutar stringa: ""String""."

Bit stringovi I

- VHDL sadrži vrednosti koje predstavljaju bitove (binarne cifre), koje mogu biti '0' ili '1'
- Bit-stirng literal predstavlja sekvencu ovih bit vrednosti
- On je predstavljen stringom cifara, okruženim dvostrukim znacima navoda, kome prethodi karakter koji određuje osnovu broja
- Oznaka za osnovu može biti neka od sledećih:
  - B za binarnu osnovu
  - O za oktalnu osnovu (osnova 8)
  - X za heksadecimalnu osnovu (osnova 16)
- Na primer, neki bit-stringovi označeni kao binarni su:

B"0100011" B"10" B"1111_0010_0001" B""

Bit stringovi II

- Možemo uključiti karaktere '_' unutar bit-string literala da bi smo razdvojili susedne cifre
- Ovim ne menjamo smisao literala, već ga samo činimo čitljivijim
- Oznaka baze može biti zapisana malim ili velikim slovom.
- Ako je oznaka za bazu (osnovu) oktalna, koriste se samo cifre od '0' do '7,
- Svaka od cifara predstavlja tačno tri bita u nizu. Na primer:

```
O"372"      -- ekvivalentno sa B"011_111_010"
o"00"       -- ekvivalentno sa B"000_000"
```

- Ako je osnova heksadecimalna, koriste se cifre '0' do '9' i karakteri 'A' do 'F' ili 'a' do 'f' (predstavljaju cifre 10 do 15). Ovde svaka cifra predstavlja četiri bita.

```
X"FA"       -- ekvivalentno sa B"1111_1010"
x"0d"       -- ekvivalentno sa B"0000_1101"
```

Konstante, promenljive i signali

Uvod I

- *Objekat* predstavlja imenovani predmet u VHDL modelu koji ima vrednost odgovarajućeg tipa

- U VHDL jeziku postoje četiri klase objekata:
  - konstante,
  - promenljive,
  - signali i
  - datoteke

- U ovom predavanju biće uvedene konstante, promenljive i signali

- Datotekama će biti posvećeno posebno predavanje

Uvod II

- Konstante, promenljive i signali su objekti koji služe za čuvanje podataka koji se upotrebljavaju unutar VHDL modela

- Razlika među njima jeste u tome što se vrednost konstante ne može menjati nakon što se ona kreira,

- Vrednost promenljive i signala može se menjati proizvoljan broj puta pomoću naredbi *dodele vrednosti promenljivoj*, odnosno naredbi *dodele vrednosti signalu*

Deklaracije konstanti I

- Konstante i promenljive moraju se deklarisati pre korišćenja u modelu
- Deklaracija uvodi ime za objekat, definiše njegov tip i može navesti njegovu početnu vrednost
- Sintaksa za deklaraciju konstante je

  deklaracija_konstante ⇐ **constant** identifikator {, . . .}: indikator_tipa [:= izraz];
- Identifikatori predstavljaju imena konstanti koja se definišu (po jedan identifikator za jednu konstantu), a indikator tipa određuje tip kome pripadaju sve konstante.
- Opcioni deo predstavlja izraz koji određuje vrednost koju svaka konstanta preuzima
- U većini slučajeva ovaj deo deklaracije je takođe neophodan

Deklaracije konstanti II

- Primeri deklaracija konstanti:

```
constant number_of_bytes: integer := 4;
constant number_of_bits: integer := 8*number_of_bytes;
constant e: real := 2.718281828;
constant prop_delay: time := 3 ns;
constant size_limit, count_limit: integer := 255;
```

- Razlog zbog čega se koriste konstante jeste mogućnost imanja imena i definisanog tipa za vrednost, umesto korišćenja direktno te vrednosti

- Ovo čini model mnogo razumljivijim za korisnika, jer ime i tip nose daleko više informacija o namenjenoj upotrebi (funkciji) objekata, od gole vrednosti

- Takođe, ukoliko prilikom razvoja modela postoji potreba za promenom vrednosti konstante, potrebno je samo ažurirati njenu deklaraciju

Deklaracije promenljivih

- Sintaksa za deklaraciju promenljive slična je deklaraciji konstante:

  deklaracija_promenljive ⇐ **variable** identifikator {, . . .}: indikator_tipa [:= izraz];

- I ovde je inicijalizacioni izraz opcion
- Ako se izostavi, promenljiva kao početnu vrednost uzima krajnju levu vrednost odgovarajućeg tipa
- Na primer, za celobrojni tip krajnja leva vrednost jeste najmanji celi broj koji može da se predstavi
- Nekoliko primera deklaracija promenljivih su:

  variable index: integer := 0;

  variable sum, average, largest: real;

  variable start, finish: time := 0 ns;

Deklaracije signala

- Kada moramo da koristiti unutrašnje signale u arhitekturnom telu, moramo ih definisati koristeći deklaracije signala

- Sintaksno pravilo za deklaraciju signala vrlo je slično onome za deklaraciju promenljivih:

  deklaracija_signala ⇐ **signal** identifikator {, . . .}: identifikator_podtipa[:= izraz];

- Deklaracija imenuje svaki signal, određuje njegov tip i opciono obezbeđuje početnu vrednost za sve signale deklarisane naredbom

- Primeri deklaracije signala

  signal index: integer := 0;

  signal sum, average, largest: real;

  signal start, finish: time := 0 ns;

Deklaracije signala

- Kada moramo da koristiti unutrašnje signale u arhitekturnom telu, moramo ih definisati koristeći deklaracije signala

- Sintaksno pravilo za deklaraciju signala vrlo je slično onome za deklaraciju promenljivih:

  deklaracija_signala ⇐ **signal** identifikator {, . . .}: identifikator_podtipa[:= izraz];

- Deklaracija imenuje svaki signal, određuje njegov tip i opciono obezbeđuje početnu vrednost za sve signale deklarisane naredbom

- Primeri deklaracije signala

  signal index: integer := 0;

  signal sum, average, largest: real;

  signal start, finish: time := 0 ns;

Dodela vrednosti promenljivoj (*variable assignment*)

- Jednom kada je promenljiva deklarisana, njena vrednost može se menjati pomoću naredbe za dodelu vrednosti promenljivoj (variable assignment statement)

naredba_dodele ⇐ [labela:] ime:=izraz;

- Opciona labela predstavlja način za identifikaciju naredbe dodele
- Ime označava promenljivu čija se vrednost menja, a izraz se izračunava da bi se došlo do nove vrednosti koja se dodeljuje promenljivoj
- Tip te nove vrednosti mora se poklapati sa tipom promenljive. Primeri:

```
program_counter := 0;
index := index+1;
```

- Prva naredba dodele postavlja vrednost promenljive *program_couter* na nulu, a druga naredba inkrementuje vrednost promenljive *index* za jedan

Dodela vrednosti signalu (*signal assignment*)

- Jednom kada je promenljiva deklarisana, njena vrednost može se menjati pomoću naredbe za dodelu vrednosti promenljivoj (variable assignment statement)

naredba_dodele ⇐ [labela:] ime<=izraz;

- Opciona labela predstavlja način za identifikaciju naredbe dodele
- Ime označava signal čija se vrednost menja, a izraz se izračunava da bi se došlo do nove vrednosti koja se dodeljuje signalu
- Tip te nove vrednosti mora se poklapati sa tipom signala. Primeri:

```
program_counter <= 0;
index <= index+1;
```

- Prva naredba dodele postavlja vrednost signala *program_couter* na nulu, a druga naredba inkrementuje vrednost signala *index* za jedan

Razlike između dodele vrednosti promenljivoj i signalu

- Važno je naglasiti razliku između naredbe za dodelu vrednosti promeljivoj i naredbe za dodelu vrednosti signalu
- Naredba za dodelu vrednosti promenljivoj odmah prepisuje novu vrednost promenljive preko stare vrednosti
- Naredba za dodelu vrednosti signalu, sa druge strane, planira dodelu nove vrednosti signalu za neki kasniji trenutak
- Detaljnije o ovome će biti reči u jednom od narednih predavanja
- Zbog ove bitne razlike između ove dve vrste dodela, VHDL koristi različite simbole za njihovo označavanje: ‘:=’ za dodelu vrednosti promenljivoj i ‘<=’ za dodelu vrednosti signalu

Skalarni tipovi

Uvod

- Koncept tipa je vrlo bitan prilikom opisivanja podataka u VHDL modelu
- Tip objekta podataka definiše skup vrednosti koje promenljiva može uzeti, kao i skup operacija koje se mogu primeniti nad tim vrednostima
- Skalarni tip sastoji se od jedne, nedeljive vrednosti
- VHDL je jezik sa jakim tipovima (strongly typed language), što znači da svaki objekat može uzeti samo vrednost tipa kome pripada
- Dalje, definicija svake operacije uključuje tipove objekata na koje se ta operacija može primeniti
- Svrha svega ovoga je da omogući rano otkrivanje grešaka, još u fazi analize modela

Deklaracije tipova

- Pomoću deklaracije tipova uvode se novi tipovi podataka u VHDL

- Deklaracija imenuje tip i određuje koje vrednosti može uzeti objekat tog tipa. Sintaksa za deklaraciju tipa je

  deklaracija_tipa ⇐ **type** identifikator **is** definicija_tipa;

- Bitna stvar koju je potrebno naglasiti je da ako se dva tipa deklarišu odvojeno sa identičnim definicijama tipa, oni su bez obzira različiti i nekompatibilni tipovi

- Na primer, ako imamo dve deklaracije tipa:

  type apples **is range** 0 **to** 100;

  type oranges **is range** 0 **to** 100;

- ne možemo dodeliti vrednost tipa *apples* promenljivoj tipa *oranges*, obzirom da su to dva različita tipa.

Celobrojni tipovi I

- U VHDL, celobrojni tipovi imaju vrednosti koje su celi brojevi
- Primer celobrojnog tipa je predefinisani tip *integer*, koji uključuje sve cele brojeve koji se mogu predstaviti na datom računaru
- Standard jezika VHDL zahteva da tip *integer* uključuje barem brojeve iz intervala –2,147,483,647 do +2,147,483,647 ($-2^{31}+1$ do $+2^{31}-1$), ali različite implementacije VHDL mogu povećavati ovaj opseg
- Sintaksno pravilo za definiciju celobrojnog tipa je

  definicija_celobrojnog_tipa ⇐ **range** prost_izraz (**to** | **downto**) prost_izraz

- Pomoću njega definiše se celobrojni tip koji sadrži cele brojeve između (i uključujući) vrednosti date sa dva prosta izraza

Celobrojni tipovi II

- Izrazi moraju da vrate celobrojne vrednosti prilikom njihovog izračunavanja
- Ako se koristi ključna reč ***to*** definišemo rastući opseg, kod kojeg su vrednosti poređane od najmanje ka najvećoj
- Ako se koristi ključna reč ***downto*** definiše se opadajući opseg, kod kojeg su vrednosti poređane sa leva na desno od najveće do najmanje. Primer:

```
type day_of_month is range 0 to 31;
type year is range 0 to 2100;
```

- Ova dva tipa su različita, iako imaju zajedničkih vrednosti. Ako deklarišemo promenljive

```
variable today: day_of_month := 9;
variable start_year: year := 1987;
```

- ne bi bilo moguće izvršiti sledeću dodelu: start_year := today;
- Iako je broj 9 član tipa *year*, on se tretira kao da je tipa *day_of_month*, koji je nekompatibilan sa tipom *year*.

Celobrojni tipovi III

- Ako želimo da koristimo aritmetičke izraze da odredimo granice opsega, vrednosti koje se koriste u izrazima moraju biti lokalno statičke, odnosno moraju biti poznate kada se model analizira. Na primer, možemo koristiti konstantne vrednosti u izrazima:

```
constant number_of_bits: integer := 32;
type bit_index is range 0 to number_of_bits-1;
```

- Operacije se mogu primeniti na vrednostima celobrojnog tipa uključuju poznate aritmetičke operacije:

+	sabiranje, identitet
-	oduzimanje, negacija
*	množenje
/	deljenje
mod	moduo
rem	ostatak pri deljenju
abs	absolutna vrednost
**	stepenovanje

Celobrojni tipovi IV

- Rezultat operacije je celi broj istog tipa kao i operand, odnosno operandi
- Za binarne operacije, operandi moraju biti istog tipa
- Desni operand operatora stepenovanja mora biti nenegativan ceo broj
- Identitet i negacija su unarni operatori, što znači da imaju samo jedan, desni operand
- Rezultat primene operatora identiteta ostavlja operand nepromenjenim, dok operator negacije kao rezultat daje rezultat koji dobijamo kada od nule oduzmemo operand
- Na primer sledeći izrazi daju isti rezultat:

A+(-B), A-(+B), A-B

Celobrojni tipovi V

- Operand deljenja kao rezultat daje ceo broj koji je rezultat deljenja, pri čemu je razlomljeni deo zaokružen ka nuli

- Operator ostatka kao rezultat daje ostatak prilikom deljenja levog operanda desnim

  5 **rem** 3 = 2, (-5) **rem** 3 = -2, 5 **rem** (-3) = 2, (-5) **rem** (-3) = -2

- Zagrade u primerima su obavezne, jer ih zahteva gramatika VHDL jezika.

- Moduo operator definisan je sledećom relacijom

  A=B*N+(A **mod** B), za neko N

- Na primer:

  5 **mod** 3 = 2, (-5) **mod** 3 = 1, 5 **mod** (-3) = -1, (-5) **mod** (-3) = -2

Celobrojni tipovi VI

- Kada se promenljiva deklariše kao promenljiva celobrojnog tipa, početna vrednost, ako nije drugačije određeno, jeste krajnji levi broj iz opsega tog tipa

- Za ulazne opsege, to će biti najmanja vrednost, a za opadajuće opsege to je najveća vrednost. Na primer:

```
type set_index_range is range 21 downto 11;
type mode_pos_range is range 5 to 7;
variable set_index: set_index_range;
variable mode_pos: mode_pos_range;
```

- Inicijalna vrednost za promenljivu *set_index* je 21, a za *mode_pos* je 5

- Početna vrednost za predefinisani tip integer je –2,147,483,647 ili manje, zbog toga što je ovaj tip predefinisan sa rastućim opsegom koji mora da uključi –2,147,483,647

Tipovi za predstavljanje realnih brojeva I

- Tipovi sa pokretnim zarezom se u VHDL koriste za reprezentaciju realnih brojeva. VHDL poseduje predefinisani tip *real*, koji uključuje barem interval –1,0E+38 do +1,0E+38, sa najmanje šest decimalnih cifara preciznosti

- Ovo odgovara IEEE standardu 32-bitne reprezentacije koja je uobičajena za brojeve sa pokretnim zarezom

- Sintaksno pravilo da definiciju novog tipa sa pokretnim zarezom je oblika:

  definicija_tipa_sa_pokretnim_zarezom ⇐ **range** prost_izraz (**to** | **downto**) prost_izraz;

- Ovo je slično pravilu za definiciju celobrojnih tipova, sa tom razlikom da prosti izrazi moraju kao svoj rezultat dati broj u pokretnom zarezu. Neki primeri definicija su:

  type input_level **is range** –10.0 **to** +10.0;

  type probability **is range** 0.0 **to** 1.0;

Tipovi za predstavljanje realnih brojeva II

- Operacije koje se mogu izvršiti nad tipovima sa pokretnim zarezom uključuju aritmetičke operacije sabiranja i identiteta (+), oduzimanja i negacije (-), množenja (*), deljenja (/), absolutne vrednosti (**abs**) i stepenovanja(**)

- Rezultat operacije je istog tipa kao i operand, odnosno operandi. Za binarne operacije operandi moraju biti istog tipa. Izuzetak je jedino desni operand za operaciju stepenovanja koji mora biti celobrojnog tipa.

- Promenljive i signali koji su deklarisani kao tipovi sa pokretnim zarezom imaju početnu vrednost koja je jednaka krajnjoj levoj vrednosti iz opsega, osim ako nije drugačije specificirano

- Na primer kao deklarišemo promenljivu *input_A* kao:

  variable input_A: input_level;

- njena početna vrednost biće –10.0.

Fizički tipovi I

- Preostali numerički tipovi u VHDL su fizički tipovi. Oni su namenjeni za prestavljanje fizičkih veličina kao što su dužina, vreme, masa, struja, itd.
- Definicija fizičkog tipa uključuje primarnu jedinicu mere i može uključiti određeni broj sekundarnih jedinica, koje moraju biti celobrojni umnošci primarne jedinice.
- Sintaksno pravilo za definiciju fizičkog tipa ima oblik:

definicija_fizičkog_tipa ⇐ **range** prost_izraz (**to** | **downto**) prost_izraz
units
identifikator;
{identifikator = fizički_literal;}
end units [identifikator]

- Definicija podseća na definiciju celobrojnog tipa sa dodatim delom za definiciju jedinica.
- Primarna jedinica (prvi identifikator nakon ključne reči *units*) predstavlja najmanju jedinicu koja postoji za taj fizički tip. Nakon nje mogu se definisati i sekundarne jedinice.

Fizički tipovi II

- Deklaracija fizičkog tipa koji bi predstavljao električnu otpornost mogla bi izgledati:

```
type otpornost is range 0 to 1E9
units
    ohm;
end units otpornost;
```

- Vrednosti za ovaj tip pišu se kao numerički broj praćen imenom jedinice, na primer:

5 ohm 22 ohm 471_000 ohm

- Obavezan je razmak pre navođenja imena jedinice. Ako je numerička vrednost jednaka 1 onda se ona može izostaviti. Sledeća dva zapisa predstavlja istu vrednost:

ohm 1 ohm

Fizički tipovi III

- Treba primetiti da vrednosti kao na primer –5 ohm ili 1E16 ohm nisu uključene u tip otpornost, zbog toga što vrednosti –5 i 1E16 ne pripadaju opsegu od 0 do 1E9

- Sekundarne jedinice definišu se tako što se navede koliko primarnih jedinica se sadrži u sekundarnoj jedinici.

- Na primer deklaracija tipa *otpornost* se može proširiti

```
type otpornost is range 0 to 1E9
units
    ohm;
    kohm = 1000 ohm;
    Mohm = 1000 kohm;
end units otpornost;
```

Fizički tip *Time*

- Predefinisani fizički tip *time* ima vrlo važnu ulogu u VHDL jer se koristi za specifikaciju kašnjenja signala u modelima. Njegova definicija je:

```
type time is range definisano_implementacijom
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;
```

- Primarna jedinica *fs*, zove se *rezolucioni limit* i predstavlja najmanji mogući korak prilikom simulacije modela. Vremenski intervali manji od rezolucionog limita zaokružavaju se na nulu.

Nabrojivi (enumerisani) tipovi I

- Često je prilikom pisanja modela na apstraktnom nivou korisno koristiti skup imena za kodirane vrednosti nekih signala umesto korišćenja binarnog kodiranja

- Nabrojivi tipovi podataka pružaju nam ovu mogućnost. Na primer, pretpostavimo da želimo modelovati procesor i želimo da definišemo imena za funkcijske kodove aritmetičke jedinice.

- Ovo može biti urađeno na sledeći način:

  type alu_function **is** (disable, pass, add, substract, multiply, divide);

- Ovakav tip se zove nabrojivi, jer su vrednosti nabrojane u listi

- Sintaksno pravilo za definiciju nabrojivog tipa je

  definicija_nabrojivog_tipa ⇐ ((identifikator | karakter_literal) {, . . .})

Nabrojivi (enumerisani) tipovi II

- Mora postojati barem jedna vrednost za tip, a svaka vrednost može biti ili identifikator ili karakter. Primer ovog drugog je:

```
type octal_digit is ('0', '1', '2', '3', '4', '5', '6', '7');
```

- Na osnovu dve gornje deklaracije, možemo deklarisati promenljive:

```
variable alu_op: alu_function;
variable last_digit: octal_digit := '0';
```

- i izvršiti dodelu na sledeći način:

```
alu_op := substract;
last_digit := '7';
```

- Kada se deklariše signal ili promenljiva koja je nabrojivog tipa, početna vrednost je krajnji levi element iz liste, ako nije drugačije naglašeno
- Tako je *unknown* početna vrednost za tip *logic_level*, a *dangerously_low* za tip *water_level*.

Standardni nabrojivi tipovi - Character

- Character tip je primer nabrojivog tipa koji kao svoje elemente ima i identifikatore i karakter literale

- Kao ilustraciju korišćenja *character* tipa deklarišimo promenljive:

```
variable cmd_char, terminator: character;
```

- i izvršimo dodelu vrednosti

```
cmd_char := 'p';
terminator := cr;
```

Standardni nabrojivi tipovi – Boolean I

- Jedan od najvažnijih predefinisanih tipova u VHDL je tip *boolean*

  type boolean **is** (false, true);

- Ovaj tip koristi se za predstavljanje uslovnih vrednosti koje mogu kontrolisati izvršavanje bihevijalnog modela

- Postoji veći broj operatora koje možemo primeniti na vrednosti različitih tipova da bi smo dobili *boolean* vrednosti, od kojih su najvažniji relacioni i logički operatori

- Relacioni operatori su operator jednakosti (‘=’) i nejednakosti (‘/=’) koji se mogu primeniti na operande bilo kog tipa (osim datoteka) uključujući i složene tipove.

Standardni nabrojivi tipovi – Boolean II

- Operandi moraju biti istog tipa, a rezultat je *boolean* vrednost. Na primer izrazi:

  123 = 123, ‘A’ = ‘A’, 7 ns = 7 ns

- svi daju rezultat *true*, dok izrazi:

  123 = 456, ‘A’ = ‘z’, 7 ns = 2 ns

- daju kao rezultat *false*.

- Logički operatori ***and***, ***or***, ***nand***, ***nor***, ***xor***, ***xnor*** i ***not*** kao operande zahtevaju *boolean* vrednosti, a kao rezultat daju *boolean* vrednost

Standardni nabrojivi tipovi – Bit

- Obzirom da se VHDL koristi za modelovanje digitalnih sistema, korisno je imati tip podataka koji bi predstavljao binarne vrednosti. Predefinisani nabrojivi tip *bit* omogućava upravo to. Definisan je kao

  type bit **is** (‘0’, ‘1’);

- Logičke operacijekoje su bile pomenute u vezi sa *boolean* tipom mogu se primeniti i na vrednost tipa *bit*, a kao rezultat daju vrednost tipa *bit*. Vrednost ‘0’ odgovara vrednosti *false*, a vrednost ‘1’ odgovara vrednosti *true*. Na primer

  ‘0’ **and** ‘1’ = ‘0’, ‘1’ **xor** ‘1’ = ‘0’

- Operandi moraju biti istog tipa, tako da je neispravno pisati

  ‘0’ **and** true

- Razlika između tipova *boolean* i *bit* je u tome što se *boolean* vrednosti koriste za modelovanje apstraktnih uslova, dok se *bit* vrednosti koriste za modelovanje električnih logičkih nivoa
- Tako ‘0’ predstavlja nizak logički nivo, a ‘1’ visok logički nivo

Standardni nabrojivi tipovi – Standard logic I

- VHDL je namenjen za modelovanje digitalnih sistema, pa je neophodno imati tipove koji će predstavljati digitalno kodirane vrednosti
- Predefinisani tip *bit* može se iskoristiti za ovu svrhu u apstraktnijim modelima, kada nas ne interesuju detalji električnih signala
- Međutim, kada želimo da produbimo model uključujući više detalja, moramo uzeti u obzir električna svojstva signala
- Ovo je moguće uraditi na mnogo načina, ali IEEE je definisao standard u biblioteci *std_logic_1164*

Standardni nabrojivi tipovi – Standard ulogic II

- Jedan od tipova definisanih u ovoj biblioteci je nabrojivi tip *std_ulogic*:

```
type std_ulogic is
        ('U'        -- neinicijalizovano
         'X'        -- nepoznato (jako – forcing)
         '0'        -- nula (jaka)
         '1'        -- jedinica (jaka)
         'Z'        -- visoka impedansa
         'W'        -- nepoznato (slabo – weak)
         'L'        -- nula (slaba)
         'H'        -- jedinica (slaba)
         '-');      -- nije bitno (don't care)
```

- Ovaj tip može se koristiti za predstavljanje signala aktivnih drajvera (jaki signali), signala rezistivnih izlaza kao što su pull-up i pull-down drajveri (slabi signali) ili tristejt drajvera koji imaju ražim rada visoka impedansa

Podtipovi I

- Često model sadrži objekte koji uzimaju samo vrednosti iz ograničenog intervala iz celog skupa vrednosti
- Ovakve objekte možemo predstaviti deklarišući podtip, koji definiše ograničen skup vrednosti iz osnovnog (baznog) tipa
- Uslov koji određuje koje vrednosti pripadaju podtipu zove se *ograničenje*.
- Koristeći podtip može se jasno naglasiti koje vrednosti su dozvoljene za objekte tog tipa. Sintaksno pravilo za deklaraciju podtipa glasi:

  deklaracija_podtipa ⇐ **subtype** identifikator **is** podtip_indikator;

  podtip_indikator ⇐ ime [**range** prost_izraz **(to** | **downto)** prost_izraz]
- Deklaracija podtipa definiše identifikator kao podtip imenovanog baznog tipa, pri čemu *range* uslov ograničava skup dozvoljenih vrednosti za podtip
- Ovaj uslov je opcioni, što znači da je moguće imati podtip koji uključuje sve vrednosti baznog tipa

Podtipovi II

- Sledeća deklaracija definiše podtip baznog tipa *integer*:

  subtype small_int **is** integer **range** –128 **to** 127;

- Vrednosti *small_int* tipa ograničene su na interval –128 do 127. Ako deklarišemo dve promenljive:

  variable deviation: small_int;

  variable adjustment: integer;

- možemo ih koristiti u izrazima, na primer: deviation := deviation+adjustment;

- Kao što se vidi iz primera, moguće je kombinovati podtipi osnovni tip u operaciji sabiranja, ali rezultat mora biti u intervalu –128 do 127, inače će biti signalizirana greška

- Sve operacije koje se mogu primeniti na osnovni tip mogu se primenjivati i na podtip tog osnovnog tipa.

Kvalifikacija podtipova (type qualification)

- Ponekad se iz konteksta ne može jasno videti kojeg je tipa konkretna vrednost
- U slučaju preopterećenih nabrojivih literala, može se javiti potreba za eksplicitnim naglašavanjem o kojem je tipu reč
- Ovo se može uraditi korišćenjem *kvalifikovanja tipova*, koje se sastoji od pisanja imena tipa praćenog jednostrukim navodnikom, a zatim izrazom u zagradama.
- Na primer ako imamo dva nabrojiva tipa

  type logic_level **is** (unknown, low, undriven, high);

  type system_state **is** (unknown, ready, busy);
- možemo razdvojiti zajedničke literale pišući

  logic_level'(unknown), system_state'(unknown)

Konverzija tipova I

- Ranije je rečeno da operandi aritmetičkih operacija moraju biti istog tipa. Ova činjenica nam ne dozvoljava mešanje operanada celobrojnog i tipa sa pokretnim zarezom.

- Ukoliko je to neophodno, možemo konvertovati jedan tip u drugi pomoću konverzije tipova

- Forma konverzije tipa je ime tipa u koji želimo da izvršimo konverziju praćeno vrednošću u zagradi

- Na primer, ako želimo izvršiti konverziju između tipova *rela* i *integer* to radimo na sledeći način

real(123), integer(3.6)

Konverzija tipova II

- Prilikom konverzije može doći do gubitka preciznosit (tačnosti)
- Konverzija tipa sa pokretnim zarezmo u tip *integer* uključuje zaokruživanje ka najbližem celom broju
- Konverzije numeričkih tipova nisu jedine koje su dozvoljene
- U opštem slučaju možemo izvršiti konverziju između sličnih tipova
- Jedna stvar na koju treba obratiti pažnju jeste razlika između kvalifikacije tipa i konverzije tipa
- Prva prosto naglašava tip vrednosti, dok druga menja vrednost, verovatno u drugi tip

Atributi skalarnih tipova

Atributi skalarnih tipova I

- Tip definiše skup vrednosti i skup dozvoljenih operacija
- U VHDL-u takođe postoji i predefinisan skup atributa koji se koriste za dobijanje informacija o vrednostima koje su uključene u tip
- Atributi se pišu posle imena tipa i jednostrukog navoda
- Vrednosti atributa mogu se koristiti za računanja u modelu

Atributi skalarnih tipova II

- Postoje atributi koji se mogu primeniti na sve skalarne tipove

- Ako sa T označimo bilo koji skalarni tip ili podtip, sa *x* vrednost tog tipa i sa *s* proizvoljni string, atributi su

T'left	prva (krajnja leva) vrednost u T
T'right	poslednja (krajnja desno) vrednost u T
T'low	najmanja vrednost u T
T'high	najveća vrednost u T
T'ascending	*true* ako T ima rastući poredak, *false* inače
T'image(x)	string koji predstavlja vrednost *x*
T'value(s)	vrednost u T koja je predstavljena sa *s*

Atributi skalarnih tipova III

- Da bi smo ilustrovali korišćenje atributa navedenih malopre, iskoristićemo sledeće deklaracije:

```
type resistance is range 0 to 1E9
units
    ohm;
    kohm = 1000 ohm;
    Mohm = 1000 kohm;
end units resistance;

type set_index_range is range 21 downto 11;

type logic_level is (unknown, low, undriven,
                     high);
```

```
resistance'left = 0 ohm
resistance'right = 1E9 ohm
resistance'low = 0 ohm
resistance'high = 1E9 ohm
resistance'ascending = true
resistance'image (2 kohm) = "2000 ohm"
resistance'value ("5 Mohm") = 5_000_000 ohm

set_index_range'left = 21
set_index_range'right = 11
set_index_range'low = 11
set_index_range'high = 21
set_index_range'ascending = false
set_index_range'image (14) = "14"
set_index_range'value ("20") = 20

logic_level'left = unknown
logic_level'right = high
logic_level'low = unknown
logic_level'high = high
logic_level'ascendnig = true
logic_level'image (undriven) = "undriven"
logic_level'value ("Low") = low
```

Atributi skalarnih tipova IV

- Dalje, postoje atributi koji se mogu primeniti samo na diskretne i fizičke tipove.

- Za bilo koji tip T, vrednost *x* i celi broj *n*, atributi su

T'pos(x)	broj pozicije *x* u T
T'val(n)	vrednost iz T na poziciji *n*
T'succ(x)	vrednost iz T na poziciji za jedan većoj od one na kojoj je *x*
T'pred(x)	vrednost iz T na poziciji za jedan manjoj od one na kojoj je *x*
T'leftof(x)	vrednost iz T na poziciji levo od pozicije *x*
T'rightof(x)	vrednost iz T na poziciji desno od pozicije *x*

Atributi skalarnih tipova V

- Za nabrojive tipove, brojevi pozicija počinju od nule za prvi element iz liste i povećavaju se za jedan za svaki naredni element

- Tako za tip *logic_level* iz prethodnog primera imamo

  logic_level'pos(unknown) = 0

  logic_level'val(3) = high

  logic_level'succ(unknown) = low

  logic_level'pred(undriven) = low

- Za celobrojne tipove, broj pozicije isti je kao i celobrojna vrednost, ali je tip vrednosti pozicije poseban tip *universal integer*

- Za fizičke tipove broj pozicije je celi broj osnovnih jedinica koje se sadrže u posmatranoj vrednosti. Na primer

  time'pos(4 ns) = 4_000_000

- obzirom da je osnovna jedinica fs.

Izrazi i operatori

Izrazi

- Pod izrazom možemo zamisliti formulu koja specificira kako treba izračunati neku vrednost

- Kao takav, izraz se sasatoji iz primarnih vrednosti povezanih pomoću operatora.

- Primarne vrednosti koje se mogu koristiti u izrazima uključuju
  - vrednosti literala
  - identifikatore koji predstavljaju objekte podataka (konstante, promenljive, itd.)
  - atribute koji daju vrednosti
  - kvalifikovane izraze
  - izraze kojima je izvršena konverzija tipa
  - izraze u zagradama

VHDL operatori I

- Svi VHDL operatori grupisani su u sledeće dve tabele
- Operatori su grupisani prema prioritetu, gde **, *abs* i *not* imaju najviši prioritet, a logički operatori najniži prioritet

Operator	Operacija	Tip levog operanda	Tip desnog operanda	Tip rezultata
**	eksponenciranje	celobrojni ili sa pokretnim zarezom	celobrojni	isti kao levi operand
abs	absolutna vrednost	celobrojni ili sa pokretnim zarezom	numerički	isti kao operand
not	negacija	celobrojni ili sa pokretnim zarezom	bit, boolean ili 1-D niz tipa bit ili boolean	isti kao operand
*	množenje	celobrojni ili sa pokretnim zarezom	isti kao levi operand	isti kao operandi
		fizički	celobrojni ili sa pokretnim zarezom	isti kao levi operand
		celobrojni ili sa pokretnim zarezom	fizički	isti kao desni operand
/	deljenje	celobrojni ili sa pokretnim zarezom	isti kao levi operand	isti kao operandi
		fizički	celobrojni ili sa pokretnim zarezom	isti kao levi operand
		fizički	isti kao levi operand	univezalni celobrojni tip
mod	moduo	celobrojni	isti kao levi operand	isti kao operandi
rem	ostatak	celobrojni	isti kao levi operand	isti kao operandi
+	identitet		numerički	isti kao operand
-	negacija		numerički	isti kao operand
+	sabiranje	numerički	isti kao levi operand	isti kao operandi
-	Oduzimanje	numerički	isti kao levi operand	isti kao operandi
&	konkatenacija	1-D niz	isti kao levi operand	isti kao operandi
		1-D niz	element tipa kojeg je levi operand	isti kao levi operand
		element tipa kojeg je desni operand	1-D niz	isti kao desni operand
		element tipa kojeg je rezultat	element tipa kojeg je rezultat	1-D niz

VHDL operatori II

Operator	Operacija	Tip levog operanda	Tip desnog operanda	Tip rezultata
sll	logičko pomeranje ulevo	1-D niz tipa bit ili boolean	celobrojni	isti kao levi operand
srl	logičko pomeranje udesno			
sla	aritmetičko pomeranje ulevo			
sra	aritmetičko pomeranje udesno			
rol	rotacija ulevo			
ror	rotacija udesno			
=	jednakost	svi osim file	isti kao levi operand	boolean
/=	nejednakost			
<	manje od	skalar ili 1-D niz bilo kog diskretnog tipa	isti kao levi operand	boolean
<=	manje ili jednako od			
>	veće od			
>=	veće ili jednako od			
and	logičko I	bit, boolean ili 1-D niz tipa bit ili boolean	isti kao levi operand	isti kao operandi
or	logičko ILI			
nand	logičko NI			
nor	logičko NILI			
xor	ekskluzivno ILI			
xnor	negacija ekskluzivnog ILI