Универзитет у Нишу

Електронски факултет

Катедра за рачунарство

Архитектура и организација рачунара

Вежбе, VHDL

Termin 3

## Case клаузула

У прошлом термину смо рекли да постоје 3 секвенцијалне наредбе које служе за управљање тока програма. IF клаузула је обрађена у претходном термину. Наредна клаузула коју ћемо обратити је **case** клаузула.

**Case** клаузулом се бира секвенца које ће се извршити зависно од вредности израза:

|  |
| --- |
| 01 **case** izraz **is**  02 **when** vrednost\_1 **=>** --vrednost - само статичке вредности (константе)  03 klauzula\_1\_1**;**  04 klauzula\_1\_2**;**  05 -- ...  06 **when** vrednost\_2 **=>** -- више вредности се повезује са |  07 klauzula\_2\_1**;**  08 klauzula\_2\_2**;**  09 -- ...  10 **when** **others** **=>**  11 klauzula\_o\_1**;**  12 -- ...  13 **end** **case** |

|  |
| --- |
| **Z** ПРИМЕР, BCD brojač, једноцифрени , броји на сваки други такт  Уведено: case; бројачи  Треба реализовати једноцифрени БЦД кружни бројач који броји унапред, на сваку другу предњу ивицу такта. |
| 01 **ENTITY** counter\_ent **IS**  02 **PORT** **(** clr **:** **IN** BIT**;**  03 clk **:** **IN** BIT**;**  04 q **:** **OUT** BIT\_VECTOR**(**3 **DOWNTO** 0**)**  05 **);**  06 **END** **ENTITY** counter\_ent**;**  07 -----------------------------------------------------  08 **ARCHITECTURE** counter\_arch **OF** counter\_ent **IS**  09 **BEGIN**  10 **PROCESS** **(**clr**,** clk**)**  11 **VARIABLE** q\_int **:** BIT\_VECTOR**(**3 **DOWNTO** 0**);**  12 **VARIABLE** cq **:** BIT**;** -- da broji svaki drugi takt  13 **BEGIN**  14 **IF** clr**=**'1' **THEN**  15 q\_int **:=** "0000"**;**  16 cq **:=** '0'**;**  17 **ELSIF** clk'**event** **and** clk**=**'1' **THEN**  18 cq **:=** **not** cq**;**  19 **IF** cq**=**'1' **THEN**  20 **CASE** q\_int **IS**  21 **WHEN** "0000" **=>** q\_int **:=** "0001"**;**  22 **WHEN** "0001" **=>** q\_int **:=** "0010"**;**  23 **WHEN** "0010" **=>** q\_int **:=** "0011"**;**  24 **WHEN** "0011" **=>** q\_int **:=** "0100"**;**  25 **WHEN** "0100" **=>** q\_int **:=** "0101"**;**  26 **WHEN** "0101" **=>** q\_int **:=** "0110"**;**  27 **WHEN** "0110" **=>** q\_int **:=** "0111"**;**  28 **WHEN** "0111" **=>** q\_int **:=** "1000"**;**  29 **WHEN** "1000" **=>** q\_int **:=** "1001"**;**  30 **WHEN** **OTHERS** **=>** q\_int **:=** "0000"**;**  31 **END** **CASE;**  32 **END** **IF;**  33 **END** **IF;**  34 q **<=** q\_int**;**  35 **END** **PROCESS;**  36 **END** counter\_arch**;** |
| Кружни бројачи броје ”у круг” - након последње вредности поново пролази кроз прву вредност. У овом случају једноцифрени БЦД бројач броји од 0 до 9 (и у круг). Бројачи се увек пројектују са неком врстом ресетовања. У овом примеру, бројач има цлр порт који доводи бројач у почетно стање (у овом примеру почетно стање је изабрано да буде 0)  За размишљање:  Да ли је овде ресет синхрони или асинхрони?  л. 11: променљивом q\_int се моделују меморијски елементи који ће памтити стање бројача. Не сме се иницијализовати променљива на 0, јер у том случају дизајн неће бити синтетизабилан. Једини исправан начин је поставити стање на 0 у телу архитектуре на неки начин (у овом случају сигнал clr узрокује ресетовање).  За размишљање:  Чему служи променљива cq? Како би се коло понашало када она не би постојала? Установите разлику у симулатору. |
|  |

Додатне напомене везане за CASE:

* CASE-ом се морају покрити све могуће вредности које израз у CASE клаузули који се испитује може имати. Због тога, CASE се мора завршити са OTHERS ако нису сви изрази покривени. OTHERS увек мора бити последњи, јер CASE није конкурентна наредба него секвенцијална, што значи да се израз секвенцијално извршавају и упоређују.
* Константе у WHEN делу CASE клаузуле могу бити и агрегати

## Loop

Наредна секвенцијална клаузула којом се управља током секвенцијалног описа **LOOP**. Postoji nekoliko tipova **LOOP** клаузула: обична LOOP, **WHILE** i **FOR LOOP** петља. Кренућемо од обичне LOOP петље.

### LOOP

**Обична LOOP се састоји од следеће синтаксе:**

|  |
| --- |
| 1 **LOOP**  2 sekvencijalni izraz -- bilo koji niz sekvencijalnih klauzula  3 **END** **LOOP;** |

Може се запазити да се овако написана петља никада неће завршити.

За размишљање:

Замислимо да постоји процес који у телу процеса има само loop клаузулу (са било којим садржајем унутар loop. Да ли би оваква конструкција имала смисла: а) синтаксно, б) семантички?

За контролу над loop клаузулом, уведене су клаузуле **exit i next**. Обе се могу писати самостално или у комбинацији са **when**:

|  |
| --- |
| 1 **EXIT;** -- kraj izvršenja loop klauzule  2 **EXIT** **WHEN** uslov -- kraj izvršenja ako je uslov ispunjen  3 **NEXT** **WHEN** uslov -- prekida tekuću iteraciju i prelazi na iduću (slično kao continue u c/c++-u) |

На крају, loop клаузуле се могу обележавати лабелама. У овом случају, име се мора написати и на крају:

|  |
| --- |
| 1 loop\_petlja**:** **LOOP**  2 statements  3 **END** **LOOP** loop\_petlja |

Лабеле се могу користити да се са **exit** или **when** изађе из било које loop клаузуле у хијерархији угњеждених клаузула:

**exit** ime\_labele **when** uslov

Пример прекидања спољње петље из угњеждене петље:

|  |
| --- |
| 1 petlja\_1**:** **loop**  2 petlja\_2**:** **loop**  3 -- some code  4 **exit** petlja\_1 **when** sigA**=**7;  5 **end** **loop**  6 **end** **loop** |
| У линији 4 се излази из главне петље под условом да је sigA=7. |

|  |
| --- |
| **Z** ПРИМЕР, бројач основе 16 са асинхроним reset улазом  Уведено: loop, exit, нумерички типови, wait until  Треба реализовати бројач основе 16. |
| 01 **ENTITY** counter **IS**  02 **PORT** **(** clk**,** reset**:** **IN** bit**;**  03 count **:** **OUT** natural**);** -- podtip od integer  04 **END** **ENTITY** counter**;**  05 ------------------------------------------  06 ------------------------------------------  07 ------------------------------------------  08 **ARCHITECTURE** behavior **OF** counter **IS**  09 **BEGIN**  10 incrementer**:**  11 **PROCESS** **IS**  12 **VARIABLE** count\_value **:** natural **:=** 0**;**  13 -- zbog inicijalizacije, nije sintetizabilno  14 **BEGIN**  15 count **<=** count\_value**;**  16 -- zbog inicijalizacije, inače za sintetizabilnu varijantu  17 -- ovo nije potrebno. Realna kola treba da se resetuju  18 -- po uključivanju pogodnom vrednošću na reset portovima.  19 -- Inače im je nedefinisano stanje.  20 **LOOP**  21 **LOOP**  22 **WAIT** **UNTIL** clk **=** '1' **or** reset **=** '1'**;**  23 **EXIT** **WHEN** reset **=** '1'**;**  24 count\_value **:=** **(**count\_value **+** 1**)** **mod** 16**;**  25 count **<=** count\_value**;**  26 **END** **LOOP;**  27 -- ovde dolazi od exit iz l. 23 kad je reset='1'  28 count\_value **:=** 0**;**  29 count **<=** count\_value**;**  30 **WAIT** **UNTIL** reset **=** '0'**;**  31 **END** **LOOP;**  32 **END** **PROCESS** incrementer**;**  33 **END** **ARCHITECTURE** behavior**;** |
| У овом примеру бројач је реализован loop клаузулама. У унутрашњој loop се налази логика самог бројања, док се у спољашњој for петљи налази логика за ресет бројача.  л. 23: процес се буди у л. 22 из једног од два разлога. Ако је разлог буђења процеса био активан reset, у овој линиј се излази из унутрашње loop, а тамо се ресетује бројач.  И у овој имплементацији бројача је променљива употребљена за моделовање меморијског елемента који чува стање бројача. Да би се свака промена бројача видела и на излазу, након сваке измене стања бројача вредност бројања се додељује излазном порту count.  Додатно запазити линије 22 и 30 gde je upotrebljena **WAIT** клаузула. У њој се сада не налази листа сигнала који се чекају, него се налази услов који се чека. Док год је тај услов неиспуњен процесс није активан.  За размишљање:  Да ли се овај бројач окида ивицом или нивоом? Шта ће се десити уколико у току прве полупериоде клока (док је клок 1) ресет пређе са 0 на 1 па назад на 0? Уочите да wаит унтил у ствари испитује услов само након догађаја на сигналима који чине услов.  Може ли без променљиве, зашто се вредност не чува у сигналу count? Који се проблем појављује у том случају? |

### FOR LOOP

FOR LOOP уводи бројач (итератор), као код бројачких петљи у програмским језицима:

|  |
| --- |
| 1 **for** i **in** diskretni\_opseg **loop**  2 klauzula\_1**;**  3 klauzula\_2**;**  4 -- ...  5 **end** **loop** |
| Променљива i се назива loop параметар, имплицитно се декларише својим навођењем, и њена вредност је аутоматски контролисана у току петље; унутар тела петље се сматра константом а ван петље није видљива. |

Дискретни опсег loop параметра се може дефинисати на више начина:

* **(\_, \_, \_, \_)** где се редом почевши од прве узимају вредности из наведене листе,
* **\_ to \_** где је представљен опсег од/до,
* **\_ downto \_** уколико је прва граница опсега већа од друге, мора се писати downto
* навођењем типа набрајања...

По свему другом, for loop се понаша исто као и loop.

|  |
| --- |
| **Z** ПРИМЕР, регистар са паралелним уписом и серијским излазом.  Уведено: GENERIC MAP, for loop, специфичности (багови) са wаит until  Tреба реализовати регистар у који се уписује вредност паралелно када је улазни порт WR=1. Након што WR постане 0, од следећег клока, на сваки клок по један бит се прослеђује на серијски излаз, почев од бита највеће тежине. После прослеђивања бита најмање тежине, серијски излаз прелази у HiZ, до следећег уписа. Док се сви битови не проследе на излаз, регистар не прима вредности иако се сигнализира упис. Креирати и тестбенч који ће да побуђује реализовани регистар. |
| 01 **ENTITY** parallel\_to\_serial **IS**  02 **GENERIC** **(**n **:** integer **:=** 8**);**  03 **PORT** **(**  04 wr**,**clk**:** **IN** std\_logic**;**  05 d\_in**:** **IN** std\_logic\_vector**(**n**-**1 **DOWNTO** 0**);**  06 d\_out**:** **OUT** std\_logic**);**  07 **END** **ENTITY** parallel\_to\_serial**;**  08 --------------------------------------------------------  09 --------------------------------------------------------  10 --------------------------------------------------------  11 **ARCHITECTURE** beh **OF** parallel\_to\_serial **IS**  12 **BEGIN**  13 **PROCESS** **IS**  14 **VARIABLE** int\_storage**:** std\_logic\_vector**(**n**-**1 **DOWNTO** 0**);**  15 **BEGIN**  16 **WAIT** **UNTIL** wr**=**'1'**;**  17 -- čeka na događaj na wr nakon koga će wr postati 1.  18 -- Bug: ako je wr stalno na 1, neće se više ništa upisivati!  19  20 int\_storage**:=**d\_in**;**  21 **FOR** i **IN** n**-**1 **DOWNTO** 0 **LOOP**  22 **WAIT** **UNTIL** clk'**event** **and** clk**=**'1'**;**  23 d\_out**<=**int\_storage**(**i**);**  24 **END** **LOOP;**  25 **WAIT** **UNTIL** clk'**event** **and** clk**=**'1'**;** -- od sledećeg kloka HiZ  26 d\_out**<=**'Z'**;**  27 **END** **PROCESS;**  28 **END** **ARCHITECTURE** beh**;**  29 --------------------------------------------------------  30 --------------------------------------------------------  31 --------------------------------------------------------  32 **ENTITY** parallel\_to\_serial\_tb **IS**  33 **GENERIC(**width **:** integer **:=** 4**);**  34 **END** **;**  35 --------------------------------------------------------  36 --------------------------------------------------------  37 --------------------------------------------------------  38 **ARCHITECTURE** parallel\_to\_serial\_tb\_arch **OF** parallel\_to\_serial\_tb **IS**  39 **SIGNAL** wr **:** std\_logic**;**  40 **SIGNAL** d\_in **:** std\_logic\_vector **(**width **-** 1 **downto** 0**);**  41 **SIGNAL** clk **:** std\_logic **:=** '0'**;**  42 **SIGNAL** d\_out **:** std\_logic**;**  43  44 **BEGIN**  45 DUT**:**  46 **ENTITY** work**.**parallel\_to\_serial**(**beh**)**  47 **GENERIC** **MAP** **(**  48 n **=>** width  49 **)** -- GENERIC MAP  50 **PORT** **MAP** **(**  51 wr **=>** wr**,**  52 d\_in **=>** d\_in**,**  53 clk **=>** clk**,**  54 d\_out **=>** d\_out  55 **);** -- PORT MAP  56  57  58 clk **<=** **not** clk **after** 50 ns**;**  59  60 stimuli**:** **process**  61 **BEGIN**  62 d\_in **<=** "0101"**;**  63 wr**<=**'1'**;**  64 -- ali! ako se promeni ovde na 0 a iza na 1,  65 -- zato što nema promena na wr ni nakon 600 ns,  66 -- nema novih upisa! to je zbog bug-a opisanog ranije  67 **WAIT** **FOR** 50 ns**;**  68 wr**<=**'0'**;**  69 **WAIT** **FOR** 600 ns**;**  70 d\_in **<=** "1101"**;**  71 wr**<=**'1'**;**  72 **WAIT** **FOR** 100 ns**;**  73 d\_in**<=**"0000"**;**  74 **WAIT** **FOR** 500 ns**;**  75 **END** **PROCESS** stimuli**;**  76 **END** **;** |
| Запазити употребу wait until клаузуле унутар тела фор loop клаузуле. Додатно запазити да сe wait у овом процесу налази на почетку, што знали да се чека да се wr сигнал са 0 промени на 1 да би се започело слање података. Када се слање података заврши, излаз parallel\_to\_serial добија стање високе импедансе.  Додатно запазити употребу GENERIC клаузуле. Константа n је употребљена да би се дефинисао регистар са бројим битова који се одређује у свакој инстанци овог регистара. Тако је у тестбенчу вредносt n предефинисана са 8 (колико је одређена у ентитету регистра) на 4, generic map клаузулом у л.48, и биће инстанциран 4-битни регистар. Коришћење GENERIC константе у границама петље је омогућило да ће регистар да се понаша коректно за било коју ширину.  Iмати у виду да се инстанца компоненте, конкурентна клаузула доделе вредности сигналу clk и proces КОНКУРЕНТНО ИЗВРШАВАЈУ.  За размишљање:  На који начин је постигнуто да је паралелни упис забрањен све док се сви битови не проследе на излаз?  У ком тренутку симулације се прихвата вредност задата у л. 73? |

### WHILE LOOP

|  |
| --- |
| 1 labela**:** **while** izraz **loop**  2 klauzula\_1  3 klauzula\_2  4 -- ...  5 **end** **loop** **[**labela**]** |

Понашање while loop клаузуле у ХДЛ-у је исто као while петљи и у програмским језицима. Међутим, while loop није погодна за моделовање синтетизабилног хардвера: while loop је синтетизабилна само под одређеним условима, разни алати за синтезу се на различит начин “сналазе” са интерпретацијом whle loop, и често њена употреба може изазвати синтезу неконтролисано великог броја кола. Из тог разлога, у овом курсу ће употреба while loop бити избегавана.

🛈 Потешкоће са синтетизабилношћу while loop ћемо донекле приближити на следећи начин: Извршавање петљи у програмским језицима захтева утрошак времена - процесор извршава итерације сукцесивно, и што је више итерација, више ће времена бити утрошено. Са друге стране, loop клаузуле у ХДЛ-у се типично синтетизују генерисањем хардвера за сваку итерацију, више итерација ће произвести више хардвера (није увек случај, али јесте у великом броју случајева). Обзиром да while може имати између 0 и бесконачно итерација, јако је тешко синтетисати између нула и бесконачно примерака хардвера за сваку итерацију.

|  |
| --- |
| **Z** ПРИМЕР, 8b синхрони бројач са дозволом бројања  Уведено: RTL стил пројектовања секвенцијалних кола; подтипови  Треба реализовати 8b кружни бројач са асинхроним ресетом, паралелним уписом, дозволом уписа, дозволом бројања и избором смера бројања. |
| 01 **LIBRARY** IEEE**;**  02 **USE** IEEE**.**std\_logic\_1164**.ALL;**  03 ---------------------------------------  04 ---------------------------------------  05 ---------------------------------------  06 **ENTITY** counter8 **IS**  07 **PORT** **(**  08 CLK**:** **IN** STD\_LOGIC**;**  09 -- moze da bude i tipa bit, onda nam ne treba biblioteka  10 RESET**:** **IN** STD\_LOGIC**;**  11 CE**,** LOAD**,** DIR**:** **IN** STD\_LOGIC**;**  12 DIN**:** **IN** INTEGER **RANGE** 0 **TO** 255**;**  13  14 COUNT**:** **OUT** INTEGER **RANGE** 0 **TO** 255  15 **);**  16 **END** counter8**;**  17 ---------------------------------------  18 ---------------------------------------  19 ---------------------------------------  20 **ARCHITECTURE** counter8\_arch **OF** counter8 **IS**  21 **BEGIN**  22 **PROCESS** **(**CLK**,** RESET**)**  23 **VARIABLE** COUNTER**:** INTEGER **RANGE** 0 **TO** 255**;**  24 **BEGIN**  25 **IF** RESET**=**'1' **THEN**  26 COUNTER **:=** 0**;**  27 **ELSIF** CLK**=**'1' **and** CLK'**event** **THEN**  28 **IF** LOAD**=**'1' **THEN**  29 COUNTER **:=** DIN**;**  30 **ELSE**  31 **IF** CE**=**'1' **THEN**  32 **IF** DIR**=**'1' **THEN**  33 **IF** COUNTER **=**255 **THEN**  34 COUNTER **:=** 0**;**  35 **ELSE**  36 COUNTER **:=** COUNTER **+** 1**;**  37 **END** **IF;**  38 **ELSE**  39 **IF** COUNTER **=**0 **THEN**  40 COUNTER **:=** 255**;**  41 **ELSE**  42 COUNTER **:=** COUNTER **-** 1**;**  43 **END** **IF;**  44 **END** **IF;**  45 **END** **IF;**  46 **END** **IF;**  47 **END** **IF;**  48 COUNT **<=** COUNTER**;**  49 **END** **PROCESS;**  50 **END** counter8\_arch**;** |
| У овом примеру је приказан типичан RTL стил пројектовања синтетизабилних секвенцијалних кола.  л. 12: Креирањем подтипа од типа INTEGER и његовим ограничавањем на 256 вредности, у синтези ће се овај порт креирати као 8-битни. Без ограничавања опсега, не би имали контролу над ширином порта типа INTEGER. |

# Сложени типови података, кориснички типови

Композитни типови података у VHDL-у, су поља (низови) и рекорди. Ови објекти могу садржати више вредности у себи.

Поље , као и у програмским језицма, садржи више вредности које су истог типа, и може се индексирати.

У VHDL-у се могу декларисати кориснички типови. Обзиром на јаку типизацију језика, објекти различитих типова се не могу мешати без експлицитне конверзије, чак и ако су можда декларисани истоветно.

Неколико примера декларације типова и поља:

|  |
| --- |
| 01 **type** memory\_word\_type **is** **array** **(**natural **range** **<>)** **of** bit**;**  02 **type** memory\_type **is** **array** **(**natural **range** **<>)** **of** memory\_word\_type**;**  03  04 **variable** memorijska\_rec**:** memory\_word\_type**(**127 **downto** 0**);**  05 **variable** memorija**:** memory\_word\_type**(**1024 **downto** 0**)(**127 **downto** 0**);**  06 ------------------------------  07 ------------------------------  08 -------- ekvivalentno --------  09 ------------------------------  10 ------------------------------  11 **type** memory\_word\_type **is** **array** **(**127 **downto** 0**)** **of** bit**;**  12 **type** memory\_type **is** **array** **(**1024 **downto** 0**)** **of** memory\_word\_type**;**  13  14 **variable** memorijska\_rec**:** memory\_word\_type**;**  15 **variable** memorija**:** memory\_word\_type**;** |
| л. 01-02: Декларисани су кориснички типови као поља са тзв. “неограниченим ” бројем елемената (*unconstrained*). Oпсег индекса поља оваквих типова (самим тим и број елемената) се одређују при декларацији објеката оваквих типова (л. 01-05)  Oвде ћемо искористити прилико да покажемо како су дефинисани BIT i BIT\_VECTOR типови. Наредне дефиниције су исте:  **SIGNAL** a **:** **ARRAY** **(**31 **DOWNTO** 0**)** **OF** bit**;**  **SIGNAL** a**:** BIT\_VECTOR**(**31 **DOWNTO** 0**)**;  У другом примеру, л.11-15, типови су одмах декларисани са одређеним опсезима индекса.  Поља су, наравно, могла да се декларишу и без претходне дефиниције корисничких типова. |

**Слог (*record*)** представља копозитни тип података који се састоји од више елемената који могу бити различитог типа. Елемтима слога приступамо преко имена. Слог се дефинише на следећи начин:

|  |
| --- |
| 01 **type** ime\_sloga **is** **record**  02 ime\_1**:** tip\_1  03 ime\_2**:** tip\_2  04 -- ...  05 **end** **record**  06 --------------------  07 --------------------  08 --------------------  09 **type** arp\_record **is** **record**  10 ip\_adr**:** BIT\_VECTOR**(**31 **downto** 0**);**  11 mac\_adr**:** BIT\_VECTOR**(**47 **downto** 0**);**  12 valid**:** boolean**;**  13 **end** **record**  14  15 **variable** jedan\_arp\_record**:** arp\_record**;** |

Слогови се неће користити у овом курсу.

У VHDL-у такође постоје типови набрајања. Тип набрајања који постоји у VHDL-у је сличан типу набрајања из било ког другог програмског језика. Тип набрајања се дефинише тако што се сваком члану даје име. Тип набрајања представља уређен тип података, што значи да се објекти типа набрајања могу поредити. Дајемо дефиницију како писати тип набрајања и пример једног оваквог типа.

|  |
| --- |
| 1 **type** ime\_tipa **is** **(**ime\_1**,** ime\_2[**,** **…**]**)**  2 ----------------------------  3 ----------------------------  4 ----------------------------  5 **type** color **is** **(**red**,** green**,** blue**)**  6 **variable** boja**:** color **:=** red  7 **signal** S**:** color **<=** blue |

Још примера у којима се користе типови набрајања се могу наћи на адреси: <https://www.vhdl-online.de/courses/system_design/vhdl_language_and_syntax/extended_data_types/enumeration_types> .

Опсег индекса вектора може бити и типа набрајања. Нпр:

|  |
| --- |
| 01 **TYPE** controller\_state **IS** **(**initial**,** idle**,** **active,** error**);**  02 **TYPE** state\_counts **IS** **ARRAY** **(**idle **TO** error**)** **OF** NATURAL**;**  03  04 **variable** state\_counter**:** state\_counts**;**  05 state\_counter**(active)** **:=** 25**;**  06 ---------------------------------------------------  07 ---------------------------------------------------  08 ---------------------------------------------------  09 **TYPE** state\_counts **IS** **ARRAY** **(**controller\_state **RANGE** idle **TO** error**)** **OF** NATURAL**;**  10 **SUBTYPE** coeff\_ram\_address **IS** integer **RANGE** 0 **TO** 63**;**  11 **TYPE** coeff\_array **IS** **ARRAY** **(**coeff\_ram\_address**)** **OF** REAL**;**  12  13 **VARIABLE** buffer\_register**,** data\_register **:** word**;**  14 **VARIABLE** counters **:** state\_counts**;**  15 **VARIABLE** coeff **:** coef\_array**;** |

Димензије вишедимензионалних поља не морају да буду истог типа. Нпр:

|  |
| --- |
| 1 **type** symbol **is** **(**'a'**,**'f'**,**'d'**,**'h'**,**digit**,**cr**,**error**);**  2 **type** state **is** **range** 0 **to** 6**;** -- nije naveden tip, podrazumeva se bilo koji ugrađeni tip koji ima literale 0 i 6. To je integer.  3 **type** transition\_matrix **is** **array** **(**state**,** symbol**)** **of** state**;**  4 ---------------------------  5 ---------------------------  6 ---------------------------  7 **variable** transition\_table **:** transition\_matrix**;**  8 transition\_table**(**5**,**'d'**);** |
| У овом примеру, индекси прве димензије (врста) променљиве transition\_table су целобројног типа, а индекси друге димензије (колона) су набрајачког типа symbol. Сами елементи матрице су целобројног типа state.  У случају да више типова набрајања имају исте вредности, при навођењу вредности за дефинисање опсега, наводи се и име типа испред ранге:  **TYPE** controller\_state **IS** **(**initial**,** idle**,** **active,** error**);**  **TYPE** state\_counts **IS** **ARRAY** **(**controller\_state **RANGE** idle **TO** error**)** **OF** NATURAL**;**  **SUBTYPE** coeff\_ram\_address **IS** integer **RANGE** 0 **TO** 63**;**  **TYPE** coeff\_array **IS** **ARRAY** **(**coeff\_ram\_address**)** **OF** REAL**;** |

### Pristup poljima

У програмским језицима, може се приступати једном елементу поља, али у VHDL-у се може приступати и целом пољу истовремено, или једном његовом делу (slice).

|  |
| --- |
| 1 coef**(**0**)** **:=** 0.0**;** -- jednom elementu, niz indeksiran celobrojnim tipom  2 counters**(active)** **:=** counters**(active)+**1**;** -- elementu niza indeksinanog sa enum  3  4 data\_register **:=** buffer\_register**; --celom nizu se može pristupati odjednom!**  5 -- ili delovima nizova**:**  6 buffe\_register**(**31 **downto** 16**)** **:=** data\_register**(**15 **downto** 0**); --ograničenje: da su slices sa leve i desne strane dodele istih dimenzija** |

Slicing важи и за bit\_vector/std\_logic\_vector.

|  |
| --- |
| **Z** ПРИМЕР, Меморија  Uvedeno: polja (nizovi), korisnički tipovi, podtipovi  Треба реализовати модел меморије која памти 64 реалне вредности, иницијализована нулама. |
| 01 **SUBTYPE** coeff\_ram\_address **IS** integer **RANGE** 0 **TO** 63**;**  02 ------------------------------  03 ------------------------------  04 ------------------------------  05 entitiy coeff\_ram **is**  06 **port** **(** rd**,** wr**:** **in** bit**;**  07 addr**:** **in** coeff\_ram\_address**;**  08 d\_in**:** **in** real**;**  09 d\_out**:** **out** real  10 **);**  11 **end** **entity** coeff\_ram**;**  12 ------------------------------  13 ------------------------------  14 ------------------------------  15 **architecture** abstract **of** coeff\_ram **is**  16 **begin**  17 memory**:** **process** **is**  18 **type** coeff\_array **is** **array** **(**coeff\_ram\_address**)** **of** real**;**  19 **variable** coeff**:** coeff\_array**;**  20  21 **begin**  22 **for** index **in** coeff\_ram\_address **loop**  23 coef**(**index**)** **:=** 0.0**;**  24 -- inicijalizacija nije sintetizabilna!  25 -- umesto toga, reset port... kao u ranijim primerima  26 **end** **loop;**  27 **loop**  28 **wait** **on** rd**,** wr**,** addr**,** d\_in**;**  29 **if** rd **=** '1' **then**  30 d\_out **<=** coeff**(**addr**);**  31 **end** **if;**  32 **if** wr**=**'1' **then**  33 coeff**(**addrr**)** **:=** d\_in**;**  34 **end** **if;**  35 **end** **loop;**  36 **end** **process** memory**;**  37 **end** **architecture** abstract**.** |
| л. 01,18: Ове декларације типова су већ објашњаване раније, овде смо их само поновили да би пример био целовит.  Дефинисани coeff\_ram\_address подтип типа integer одређује опсег мем. адреса.  л. 19: Променљивом coeff\_array се моделује сама.  Ентитет садржи rd и wr портове (дозвола читања и уписа), d\_in и d\_out (податак који се уписује/чита) и addr за саму адресу са које се чита/уписује.  **Питања за размишљање:**  Шта се дешава када су rd и wr су заједно 1?  Шта ће се десите ако wait у л. 28 има мање сигнала (проверити у симулатору)? |