Универзитет у Нишу

Електронски факултет

Катедра за рачунарство

Архитектура и организација рачунара 2

Вежбе, VHDL

Термин 4

|  |
| --- |
| **Z** ПРИМЕР, Меморија , синхрона  Уведено : конверзија између типова integer и std\_logic\_vector  Треба написати опис меморије која може да чува 256 8-битних вредности. Упис је регулисан клоком и синхроном дозволом уписа. |
| 01 **LIBRARY** IEEE**;**  02 **USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**  03 **USE** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;**-- za conv\_integer()  04 ------------------------------------------  05 ------------------------------------------06 ------------------------------------------  07 **ENTITY** Memorija **IS**  08 **PORT** **(** WE **:** **IN** STD\_LOGIC**;**  09 clk **:** **in** STD\_LOGIC**;**  10 addr **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**  11 data **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**  12 Q **:** **out** STD\_LOGIC\_VECTOR **(**7 **downto** 0**)**  13 **);**  14 **end** Memorija**;**  15 ------------------------------------------  16 ------------------------------------------  17 ------------------------------------------  18 **architecture** Behavioral **of** Memorija **is**  19 **type** ram\_mem\_type **is** **array** **(**255 **downto** 0**)** **of** STD\_LOGIC\_VECTOR**(**7 **downto** 0**);**  20 **signal** rammem **:** ram\_mem\_type**;**  21 **begin**  22  23 **process** **(**clk**)**  24 **variable** addrtemp**:** integer **range** 255 **downto** 0**;**  25 **begin**  26 **if** **(**clk'**EVENT** **and** clk **=** '0'**)** **then**  27 addrtemp **:=** CONV\_INTEGER**(**addr**);**  28 **if** **(**WE**=** '1'**)** **then**  29 rammem**(**addrtemp**)** **<=**data**;**  30 **end** **if** **;**  31 Q**<=** rammem**(**addrtemp**);**  32 **end** **if** **;**  33 **end** **process;**  34 **end** **architecture;** |
| Изостављањем иницијализације у овом примеру, за разлику од претходног, дизајн постаје синтетизабилан. Одговорност корисника је да не чита локације у којима ништа није уписано, или да се све локације при покретању иницијализују на неку вредност. Реалне меморије поседују и ресет порт којим корисник на почетку поставља све локације на нула.  л 19, 20: Меморија је моделована низом од 256 елемената. Да би се приступало елементима низа, не може се користити вишебитни податак addr, већ се мора користити податак нумеричког типа. Због тога је у л. 27 извршена конверзија STD\_LOGIC\_VECTOR у integer вредност. Функција за конверзију CONV\_INTEGER се налази у пакету STD\_LOGIC\_UNSIGNED, који се мора укључити на почетку (л. 03).  Питања:  Којом ивицом се окида овај модул?  Порт дозволе читања овде не постоји; како се понаша овај модул, када се могу прочитати подаци?  За размишљање:  Проверити у симулацији који податак се чита са локације у коју се у том тренутку и уписује - претходни или управо уписани податак. |

# Atributi složenih tipova

Представићемо овде неколико основних атрибута сложених (композитних) типова. Ови атрибути су синтетизабилни.

Помоћу ових атрибута, могу се добити индекси или опсези индекса низова. Вредности које враћају, зависе од декларације низа на који се примењују.

Нпр. нека су дати низ и матрица:

**SIGNAL** d **:** STD\_LOGIC\_VECTOR **(**7 **DOWNTO** 0**);**

**variable** matrix **is** **ARRAY** **(**1 **to** 5**,** 6 **to** 10**)** **of** integer**;**

Примери атрибута и њихово значење у овим случајевима:

d'**LOW=**0**,** d'**HIGH=**7 најмања и највећа вредност индекса

d'**LEFT=**7**,** d'**RIGHT=**0**,** први (“леви”) и последњи (“десни”) индекс

d'**LENGTH=**8 дужина низа по декларацији

d'**RANGE=(**7 **downto** 0**)** дискретни опсег индекса, по декларацији

matrix'**LEFT(**1**)** први индекс прве димензије --> 1

matrix'**LEFT(**2**)** први индекс друге димензије --> 6

matrix'**LEFT** подразумева се прва димензија --> 1

matrix'**RIGHT** --> 5

matrix'**LENGTH** --> 5 (od 1 do 5)

matrix'**LENGTH(**2**)** --> 5 (od 6 do 10)

matrix'**RANGE(**2**)** --> 6 to 10

Атрибути могу да се користе и за грађење синтетизабилних дискретних опсега, нпр за обилазак низова:

**FOR** i **IN** d'**RANGE** **LOOP** **...**

**FOR** i **IN** **RANGE** **(**d'**LEFT** **DOWNTO** d'

**RIGHT)** **LOOP** **…** није безбедно, пошто зависи од декекларације d. У овом случају ради пошто је d декларисан такође са **DOWNTO**, (па је d'**LEFT** веће од d'**RIGHT**), иначе би била грешка.

**FOR** i **IN** **RANGE** **(**d'**LENGTH-**1 **DOWNTO** 0**)** **LOOP** **…**

**FOR** i **IN** matrix'**RANGE(**1**)**

**FOR** j **in** matrix'**RANGE(**2**)**

-- klauzule...

|  |
| --- |
| **Z** ПРИМЕР, бројање водећих нула у улазном податку  Уведено : атрибут низа за обилазак кроз низ; постизање while понашања помоћу for-loop и exit.  Треба описати комбинациону мрежу која на излазу поставља број водећих нула (непрекидне секвенце нула са стране веће тежине) у улазном вишебитном податку. |
| 01 **LIBRARY** IEEE**;**  02 **USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**  03 --------------------------------  04 --------------------------------  05 --------------------------------  06 **ENTITY** LeadingZeros **IS**  07 **GENERIC(** n **:** INTEGER **:=** 8**);**  08 **PORT** **(** --kombinaciona mreža, nema kloka!  09 data**:** **IN** STD\_LOGIC\_VECTOR **(**n**-**1 **DOWNTO** 0**);**  10 zeros**:** **OUT** INTEGER **RANGE** 0 **TO** N  11 **);**  12 **END** LeadingZeros**;**  13 --------------------------------  14 --------------------------------  15 --------------------------------  16 **ARCHITECTURE** behavior **OF** LeadingZeros **IS**  17 **BEGIN**  18 **PROCESS** **(**data**)**  19 **VARIABLE** count**:** INTEGER **RANGE** 0 **TO** n**;**  20 **BEGIN**  21 count **:=** 0**;**  22 **FOR** i **IN** data'**RANGE** **LOOP** -- broji od bita n-1 do bita 0, jer je tako deklarisan signal data  23 **CASE** data**(**i**)** **IS**  24 **WHEN** '0' **=>** count **:=** count **+** 1**;**  25 **WHEN** **OTHERS** **=>** **EXIT;**  26 **END** **CASE;**  27 **END** **LOOP;**  28 zeros **<=** count**;**  29 **END** **PROCESS;**  30 **END** behavior**;** |
| У овом примеру је искоришћен RANGE атрибут сигнала дата.  л. 25: **EXIT** uзрокује да се прекине **loop**, и тако се броје само водеће нуле. У програмским језицима је природније да се овакво функционисање опише **while** петљом, али , како је већ дискутовано, **while** треба избегавати у HDL-u. Уместо **while**, безбедно је применити технику која је овде приказана. |
| Питања:   * + Која је функција клаузуле when у л. 25? Када ће се активирати тај случај?   + Који је резултат ако је data=00010001?   + Kако би се понашало коло да је у л. 22 написано:   FOR i IN data'LOW to data'HIGH LOOP  FOR i IN data'HIGH downto data'LOW LOOP   * Како би се понашало коло да је сигнал data дефинисан са 0 **TO** n-1? * Како се може установити да је ово комбинационо коло? |

# Конкатенација

Оператор конкатенације је &.

Конкатенација је могућа над вишебитним подацима:

|  |
| --- |
| 1 "000" **&** "111" --> "000111"  2 '0' **&** "111" --> "0111"  3 "000" **&** '1' --> "0001"  4 '0' **&** '1' --> "01" |

При чему операнди могу бити литерали (као у претходним примерима), али и идентификатори - сигнали и променљиве. Конкатенација , у комбинацији са *slicing*-om омогућава широк спектар манипулације над битовима (биће приказано у примерима касније).

Конкатенација се може применити и над низовима. Нпр. ако је a низ од 8 елемената, а b и c низови од по 4 елемента (било ког типа), може:

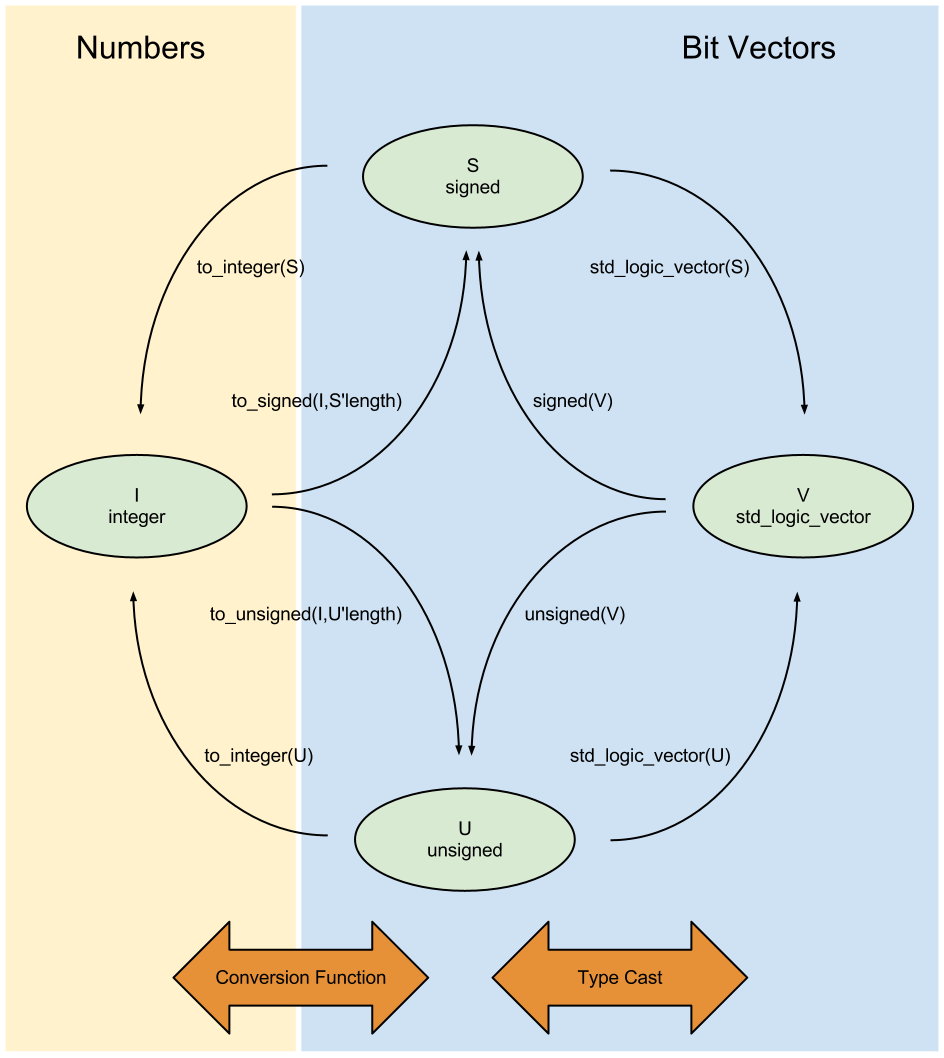
|  |
| --- |
| a**<=**b**&**c**;**  b**<=**b**&**c**;** -- nije ispravno, dužina s desne strane je 8, s leve je 4.  b**<=**b**(**0 **to** 2**)&**c**(**1**)** |

Охрабрујемо Вас да ово пробате за различите декларације.

|  |
| --- |
| **Z** ПРИМЕР Коло за кашњење  Уведено : конкатенација низова  Моделовати коло које закасни одређен број тактова податак са улаза. Написати и тестбенч |
| 01 **ENTITY** shift\_reg **IS**  02 **PORT** **(**clk **:** **IN** bit**;**  03 din **:** **IN** integer**;**  04 dout **:** **OUT** integer**);**  05 **END** **ENTITY** shift\_reg**;**  06 -------------------------------------------  07 -------------------------------------------  08 -------------------------------------------  09 **ARCHITECTURE** arch **OF** shift\_reg **IS**  10 **type** int\_array **is** **array** **(**0 **to** 3**)** **of** integer**;**  11 **SIGNAL** d**:** int\_array**;**  12 **BEGIN**  13 **PROCESS** **(**clk**)**  14 **BEGIN**  15 **if** **(**clk'**event** **and** clk**=**'1'**)** **then**  16 d**<=** d**(**1 **to** 3**)** **&** din**;**  17 -- ili, ako se izabere promenljivo kasnjenje,  18 -- ako umesto 3 bude generic konstanta n,  19 -- moze umesto 3 da se pise n ili d'high, ili d'right  20 -- (jer je 0 to 3, 'high je ovde isto sto i 'right)  21 **end** **if;**  22 **END** **PROCESS;**  23  24 dout **<=** d**(**0**);**  25 **END** arch**;**  26 -------------------------------------------  27 -------------------------------------------  28 -------------------------------------------  29 **entity** shift\_reg\_tb **is**  30 **end** shift\_reg\_tb**;**  31 -------------------------------------------  32 -------------------------------------------  33 -------------------------------------------  34 **architecture** shift\_reg\_tb\_arch **of** shift\_reg\_tb **is**  35  36 **SIGNAL** clk**:** bit **:=**'0'**;**  37 **signal** din**,** dout **:** integer**;**  38  39 **begin**  40 UUT**:** **entity** work**.**shift\_reg**(**arch**)**  41 **port** **map** **(**  42 clk **=>** clk**,**  43 din **=>** din**,**  44 dout **=>** dout  45 **);**  46  47 **PROCESS** **(**CLK**)**  48 **BEGIN**  49 clk**<=not** clk **after** 50 ps**;**  50 **END** **PROCESS;**  51  52 stimuli**:** **process** **is**  53 **begin**  54 din**<=** 1**,**  55 2 **after** 100 ps**,**  56 3 **after** 200 ps**,**  57 4 **after** 300 ps**,**  58 5 **after** 400 ps**;**  59  60 **wait** **for** 500 ps**;**  61 **end** **process** stimuli**;**  62 **end** shift\_reg\_tb\_arch**;** |
| Коло за кашњење је имплементирано као каскадни низ меморијских елемената; са сваким тактом податак се помера кроз низ.  л. 16: са десне стране, начињен је низ од 4 компоненте од три компоненте низа d, на шта је настављен din. Овај низ је додељен низу d. На овај начин су сви елементи, осим оног с индексом 0, померени за једно место у лево, а на последње место је дошао улазни порт.  Да би било јасније, ова линија је могла да се напише и на експлицитан начин: d**(**0 **to** 3**)<=** d**(**1 **to** 3**)** **&** din**;**. Ово се у ствари имплицитно разлаже на компоненте, као да је написан низ додела:  d**(**0**)<=** d**(**1**);**  d**(**1**)<=** d**(**2**);**  d**(**2**)<=** d**(**3**);**  d**(**3**)<=** din**;**  Због делта кашњења, овај низ додела у ствари имплементира померање.  л. 24: елемент низа d**(**0**)** се прослеђује на излазни порт. Како је ово конкурентна клаузула, свака нова вредност d**(**0**)** се прослеђује. (a d**(**0**)** добије нову вредност у процесу на сваки такт) |
| Питања   * Представите и објасните таласни облик dout у времену за дату побуду. |

|  |
| --- |
| **Z** ПРИМЕР Адресни декодер, портови на активно ниским нивоима на излазном порту.  Уведено: ieee.numeric\_std за конверзију целобројних типова у вишебитне типове; не -елементарни стимулуси помоћу for.  Пројектовани адресни декодер n:2^n са активно ниским нивоом на излазу и enable контролом. |
| 01 **library** ieee**;**  02 **use** ieee**.**std\_logic\_1164**.all;**  03 **use** ieee**.**numeric\_std**.all;**  04 --zbog unsigned tipa, i funkcije to\_integer  05 --use ieee.std\_logic\_arith.all;  06 -- i ovde postoji definicija unsigned, sa funkcijom za konverziju  07 -- conv\_integer  08 ---------------------------------------  09 ---------------------------------------  10 **ENTITY** test **IS**  11 **generic** **(**n **:** positive**:=**2**);**  12 -- br bitova adrese  13 -- positive je subtype od integer  14 -- alt, za l. 20: subtype output\_range IS integer range 2\*\*n-1 downto 0;  15 **PORT** **(**  16 EN **:** **IN** std\_logic**;**  17 address **:** **IN** unsigned**(**n**-**1 **downto** 0**);**  18 -- unsigned je u stvari std\_logic\_vector,  19 -- koji se moze tretirati kao neoznacen broj  20 decoded\_address **:** **OUT** std\_logic\_vector**(**2**\*\***n**-**1 **downto** 0**)** -- ako je definisan podtip iz l. 14, onda može: (output\_range), isto i u l.29  21 **);**  22 **END** **ENTITY** test**;**  23 ---------------------------------------  24 ---------------------------------------  25 ---------------------------------------  26 **ARCHITECTURE** arch **OF** test **IS**  27 **BEGIN**  28 **PROCESS** **(**EN**,** address**)**  29 **variable** internal**:** std\_logic\_vector**(**2**\*\***n**-**1 **downto** 0**);**  30 **variable** addr**:** natural **range** 0 **to** 2**\*\***n**-**1**;**  31 **BEGIN**  32 addr **:=** **to\_integer(**address**);** -- iz paketa numeric\_std  33 -- addr := conv\_integer(address); -- iz paketa std\_logic\_arith  34 internal**:=** **(others=>**'1'**);** --svi su '1'  35 **if** **(**EN**=**'1'**)** **then**  36 internal**(**addr**):=**'0'**;** --a samo jedan je ipak '0'.  37 **end** **if;**  38 decoded\_address **<=** internal**;**  39 **END** **PROCESS;**  40 **END** arch**;**  41 -- diskusija: da li je moglo bit slicing-om?  42 --internal:=(ena=>'0', others=>'1')? Ne! jer oznaka bitova mora da je  43 --konstanta, ne moze signal (kao sto je to ena)  44 ------------------  45 ---------------------------------------  46 ---------------------------------------  47 ---------------------------------------  48 **library** ieee**;**  49 **use** ieee**.**std\_logic\_1164**.all;**  50 --use ieee.std\_logic\_arith.all;  51 -- zbog unsigned  52 **use** ieee**.**numeric\_std**.all;**  53 --zbog to\_unsigned  54 ---------------------------------------  55 ---------------------------------------  56 ---------------------------------------  57 **entity** test\_tb **is**  58 **generic(** n**:** positive **:=** 4**);**  59 -- stvarni broj bitova za koji ce se testirati  60 **end** test\_tb**;**  61 ---------------------------------------  62 ---------------------------------------  63 ---------------------------------------  64 **architecture** test\_tb\_arch **of** test\_tb **is**  65 **SIGNAL** EN**:** std\_logic**;**  66 **signal** address **:** unsigned**(**n**-**1 **downto** 0**);**  67 **signal** decoded\_address **:** std\_logic\_vector**(**2**\*\***n**-**1 **downto** 0**);**  68 **begin**  69 UUT**:** **entity** work**.**test**(**arch**)**  70 **generic** **map(**n**=>**n**)**  71 **port** **map** **(**  72 EN **=>** EN**,**  73 address **=>** address**,**  74 decoded\_address **=>** decoded\_address  75 **);**  76  77 stimuli**:** **process** **is**  78 **begin**  79 EN **<=** '0'**,** '1' **after** 100 ps**;**  80 **for** i **in** integer **range** 0 **to** 2**\*\***n**-**1 **loop**  81 -- moglo je i bez "integer range",  82 -- ovako se eksplicitno odredjuje tip promenljive i  83 -- a da li je moglo "in address'range loop"? Ne! Jer bi onda bilo 3, 2, 1, i -- 0!  84 -- 'range je opseg indeksa, ne opseg vrednosti bit vektora  85 -- ali je moglo:  86 -- for i in decoded\_address'range! samo sto bi islo unazad,  87 -- jer je definisan sa downto. Zato 'range\_reverse atribut  88 **wait** **for** 100 ps**;**  89 address**<=to\_unsigned(**i**,** n**);**-- to\_unsigned konvertuje int u std\_logic\_vector širine n  90 **end** **loop;**  91 **wait** **for** 100 ps**;** -- da bi se i poslednja vrednost zadržala pre nego što se ponovo vrati na EN<='0'  92 **end** **process** stimuli**;**  93 **end** test\_tb\_arch**;** |
| Адресни декодер декодира податак на улазу у “hotbit” код на излазу - само један од 2^n излаза је активан. Како је по тексту задатка активни ниво низак (= 0), сви излази осим једног ће бити 1. Ако је EN неактиван (= 0), сви излази су неактивни (= 1).  Питања  Да ли је могуће да се процес активира а да је EN=0? Шта се тада дешава?  Да ли је овако написан testbench тестира све могуће вредности? |

У претходним примерима је приказано како конвертовати податке из целобројних у вишебитне типове и обрнуто. Преглед конверзија између нумеричких и типова вишебитних података је дат на слици 7.



*Слика 7. Начини за конверзију између нумеричких и вишебитних типова*

# Generate Клаузула

GENERATE је конкурентна клаузула која омогућава мултиплицирање инстанци комонената. Уколико треба да се напише више инстанци компонената, које се повезују на исти или сличан начин, да се не би „ручно“ понављале инстанце (што је склоно грешкама), може се послужити генерате клаузулом.

Постоје **3 типа generate клаузуле**: if-generate, case-generate i for-generate. Ми ћемо обрадити све три клаузуле.

## FOR-GENERATE

Изглед FOR-GENERATE клаузуле дат је у наставку.

|  |
| --- |
| 1 labela**:** **FOR** generate\_parametar **IN** diskretni\_opseg **GENERATE**  2 **[**blok lokalnih deklaracija  3 **BEGIN]**  4 konkurentne klauzule  5 **END** **GENERATE** [labela]**;** |
| Ефекат for-generate: За сваку вредност генерате параметра из дискретног опсега, генерише се по једна копија блока конкурентних клаузула и блока локалних декларација. Вредност генерате параметра је константна у једној копији. Локалне декларације могу да садрже све што и декларативни део архитектуре, и видљиве су само у једној копији. |

У телу for-generate се могу наћи било које конкурентне клаузуле (процеси, инстанце компонената, кокурентне доделе сигналима, друге генерате клаузуле). Ова клаузула омогућава итеративну репликацију конкурентних клаузула које су наведене унутар тела FOR-GENERATE.

Лабела, ime FOR-GENERATE структуре, је обавезна. Додатно, дискретни опсег генерате параметра мора бити познат у време елаборације (пандам превођењу у свету програмских језика). Због тога, сигнали не смеју да учествују у дефинисању дискретног опсега, већ само константе и литерали. Атрибути сигнала се могу користити јер су они динамичке константе, везани за декларације сигнала и познате су им вредности при елаборацији.

Обзиром да FOR-GENERATE структура садржи конкурентне клаузуле, то значи да се у телу једне FOR-GENERATE структуре може наћи још FOR-GENERATE структура, могу се наћи и друге GENERATE структуре (if-generate i case-generate). Користећи угњежене FOR-GENERATE структуре могу се генерисати сложеније структуре (матрице, систоличка поља, итд.).

Више о FOR-GENERATE клаузули се може наћи нпр. на: <https://www.ics.uci.edu/~jmoorkan/vhdlref/generate.html> .

|  |
| --- |
| **Z** ПРИМЕР *Carry-ripple* сабирач подесиве ширине, конкурентним клаузулама доделе вредности сигналу.  Уведено: for-generate са конкурентним клаузулама; интерни сигнали за униформисање повезивања копија у for-generate.  Пројектовати потпуни сабирач n-битних података (n - *generic* константа) помоћу *generate* клаузуле и конкурентних клаузујла доделе вредности сигналу. |
| 01 **LIBRARY** ieee**;**  02 **USE** ieee**.**std\_logic\_1164**.all;**  03 --------------------------------  04 --------------------------------  05 --------------------------------  06 **ENTITY** carry\_ripple\_adder **IS**  07 **GENERIC** **(**n**:** INTEGER **:=** 4**);**  08 **PORT** **(**  09 a**,** b**:** **IN** STD\_LOGIC\_VECTOR **(**n**-**1 **DOWNTO** 0**);**  10 cin**:** **IN** STD\_LOGIC**;**  11 s**:** **OUT** STD\_LOGIC\_VECTOR **(**n**-**1 **DOWNTO** 0**);**  12 cout**:** **OUT** STD\_LOGIC  13 **);**  14 **END** carry\_ripple\_adder**;**  15 --------------------------------  16 --------------------------------  <17 --------------------------------  18 **ARCHITECTURE** w\_generate **OF** carry\_ripple\_adder **IS**  19 **SIGNAL** c\_int**:** STD\_LOGIC\_VECTOR **(**n **DOWNTO** 0**);**  20 **BEGIN**  21 c\_int**(**0**)** **<=** cin**;**  22 G1**:** **FOR** i **IN** 0 **TO** n**-**1 **GENERATE**  23 s**(**i**)** **<=** a**(**i**)** **XOR** b**(**i**)** **XOR** c\_int**(**i**);**  24 c\_int**(**i**+**1**)** **<=** **(**a**(**i**)** **AND** b**(**i**))** **OR** **(**a**(**i**)** **AND** c\_int**(**i**))** **OR** **(**b**(**i**)** **AND** c\_int**(**i**));**  25 **END** **GENERATE;**  26 cout **<=** c\_int**(**n**);**  27 **END** **ARCHITECTURE** w\_generate**;** |
| Сабирач је реализован слично као и раније у овом материјалу, каскадним везивањем једнобитних потпуних сабирача (излазни пренос са једне позиције се доводи на улазни пренос на следећој позицији).  За прослеђивање преноса између позиција уведен је сигнал c\_int (л. 19). Он има n+1 бит, на бит 0 се доводи улазни пренос целог сабирача (л. 21), а излазни пренос последњег сабирача се доводи на бит n сигнала c\_int. Након овога, повезивање свих једнобитних сабирача се може описати на униформан начин (л. 23-24).  л. 23-34: једнобитни сабирачи су овде описани конкурентним клаузулама, уместо инстанцама компонената као у ранијем примеру. Сума и излазни пренос на једном биту су описани логичким изразима (који су познати у теорији). Могу се добити и други облици, минимизацијом прекидачких функција из табеле истинитости функција S и Cout за потпуни сабирач. |
| За размишљање:  Нацртати шему повезивања сигнала c\_int на једнобитне сабираче и портове n-битног сабирача.  Како би изгледао опис када би се, уместо конкуретних клаузула доделе вредности користиле инстанце једнобитних сабирача, дефинисаних раније у материјалу? |

|  |
| --- |
| **Z** ПРИМЕР Регистар са дозволом излаза, сачињен од Dff-ova и тростатичких бафера.  Уведено: for-generate са инстанцама компоената; локалне декларације у generate.  Пројектовати n-bitni (n - generic константа) регистар са асинхроном дозволом излаза. За имплементацију регистра користити инстанце D флипфлопова и тростатичких бафера, које такође треба описати. |
| 01 **LIBRARY** ieee**;**  02 **USE** ieee**.**std\_logic\_1164**.ALL;**  03 --------------------------------------  04 --------------------------------------  05 --------------------------------------  06 **ENTITY** D\_flipflop **IS**  07 **PORT** **(**  08 clk**,** d**:** **IN** std\_logic**;**  09 q**:** **OUT** std\_logic  10 **);**  11 **END** **ENTITY** D\_flipflop**;**  12 --------------------------------------  13 --------------------------------------  14 --------------------------------------  15 **ARCHITECTURE** simple **OF** D\_flipflop **IS**  16 **BEGIN**  17 **PROCESS(**clk**)**  18 **BEGIN**  19 **IF** clk'**event** **and** clk**=**'1' **THEN**  20 q**<=**d**;**  21 **END** **IF;**  22 **END** **PROCESS;**  23 -- ili samo q<=d when clk'event and clk='1';  24 **END** **ARCHITECTURE** simple**;**  25 --------------------------------------  26 --------------------------------------  27 --------------------------------------  28 **LIBRARY** ieee**;**  29 **USE** ieee**.**std\_logic\_1164**.ALL;**  30 --------------------------------------  31 --------------------------------------  32 --------------------------------------  33 **ENTITY** tristate\_buffer **IS**  34 **PORT** **(**  35 a**,** en**:** **IN** STD\_LOGIC**;**  36 y**:** **OUT** STD\_LOGIC  37 **);**  38 **END** **ENTITY** tristate\_buffer**;**  39 --------------------------------------  40 --------------------------------------  41 --------------------------------------  42 **ARCHITECTURE** beh **OF** tristate\_buffer **IS**  43 **BEGIN**  44 y**<=**a **WHEN** en**=**'1' **ELSE** 'Z'**;**  45 **END** **ARCHITECTURE** beh**;**  46 --------------------------------------  47 --------------------------------------  48 --------------------------------------  49 **LIBRARY** ieee**;**  50 **USE** ieee**.**std\_logic\_1164**.ALL;**  51 --------------------------------------  52 --------------------------------------  53 --------------------------------------  54 **ENTITY** register\_tristate **IS**  55 **GENERIC** **(**width**:** positive**);**  56 **PORT** **(**  57 clock**:** **IN** std\_logic**;**  58 out\_enable**:** **IN** std\_logic**;**  59 data\_in**:** **IN** std\_logic\_vector **(**0 **TO** width**-**1**);**  60 data\_out**:** **OUT** std\_logic\_vector **(**0 **TO** width**-**1**)**  61 **);**  62 **END** **ENTITY** register\_tristate**;**  63 --------------------------------------  64 --------------------------------------  65 --------------------------------------  66 **ARCHITECTURE** cell\_level **OF** register\_tristate **IS**  67 **BEGIN**  68 cell\_array**:** **FOR** bit\_index **IN** 0 **TO** width**-**1 **GENERATE**  69 **SIGNAL** data\_unbuffered**:** std\_logic**;** -- lokalni signal za svaku kopiju  70 **BEGIN** -- !! ne može se ovde kreirati signal za medusobno povezivanje kopija, to bi moralo u arhitekturi  71 cell\_storage**:** **ENTITY** work**.**D\_flipflop**(**simple**)**  72 **PORT** **MAP** **(**  73 clk**=>**clock**,**  74 d**=>**data\_in**(**bit\_index**),**  75 q**=>**data\_unbuffered  76 **);**  77 cell\_buffer**:** **ENTITY** work**.**tristate\_buffer**(**beh**)**  78 **PORT** **MAP** **(**  79 a**=>**data\_unbuffered**,**  80 en**=>**out\_enable**,**  81 y**=>**data\_out**(**bit\_index**)**  82 **);**  83 **END** **GENERATE** cell\_array**;**  84 **END** **ARCHITECTURE** cell\_level**;** |
| Ентитети D\_flipflop и tristate\_buffer се инстанцирају у регистру. Већ су разматрани раније, и нећемо их овде посебно објашњавати.  У ентитету register\_tristate, компоненте D\_flipflop и tristate\_buffer су међусобно повезани сигналом data\_unbuffered. Поново: декларација унутар декларативног дела generate клаузуле је локална за једну копију, и сваки бит регистра има по један сигнал data\_unbuffered.  Сваки бит регистра се састоји од једног D\_flipflop и једног tristate\_buffer. Портови register\_tristate ентитета би сви били дужине 1. Сигнал data\_unbuffered повезује D\_flipflopс са tristate\_buffer тако што је излаз стања флипфлопа повезан на улаз 3state бафера.  Када се клок активира, садржај који се налази на data\_in се уписује у D- FlipFlop. Када је сигнал out\_enable активан онда се оно што се налази у D FlipFlipovima прослеђује на излаз data\_out. Ако сигнал out\_enable није активан онда је излаз из register\_tristate у стању високе импедансе.  Ово коло је могло да се реализује и тако што би се прво креирао ентитет за један бит регистра: који обухвата по једну инстанцу флипфлопа и тростатичког бафера. У том случају би се тај нови ентитет инстанцирао у генерате клаузули. Строго узев, текст овог примера такву могућност не дозвољава директно, тако да би на испиту било боље у случају оваквог текста задатка не креирати нови ентитет. |
| За размишљање:  л.23: Зашто се линија из коментара не може написати уместо доделе у л.20? Шта још треба променити да би Dff могао да се опише овом линијом?  Нацртати шему повезивања флипфлопа и тростатичког бафера која је реализована у овом примеру.  Како би се могло реализовати ово коло без локалних декларација у generate клаузули? |

## IF-GENERATE и CASE-GENERATE

IF-GENERATE клаузула кондиционо укључује или искључује неке конкурентне клаузуле. Изглед IF-GENERATE клаузуле је дат у наредном блоку.

|  |
| --- |
| 01 labela**:** **IF** uslov\_0 **GENERATE**  02 **[**blok lokalnih deklaracija  03 **BEGIN]**  04 konkurentne klauzule  05 **ELSIF** uslov\_1 **GENERATE**  06 konkurentne klauzule  07 -- ...  08 **ELSE** uslov\_n **GENERATE**  09 konkurentne klauzule  10 **END** **GENERATE** labela**;** |

IF-GENERATE добро допуњује FOR-GENERATE клаузулу: обично су крајње копије у for-generate (прва и последња) другачије повезане од копија “у средини”. У овим случајевима, IF-GENERATE клаузула може да има услов да су вредности generate параметра једнаке крањим, и онда ће се инстанцирати другачије копије од осталих.

Исто као и код FOR-GENERATE клаузуле, вредности услова морају бити познате у фази елаборације.

Уколико постоји потреба да се користе сигнали и променљиве у условима, то се не може реализовати у generate клаузули. Онда се дизајн мора променити да се користе секвенцијалне клаузуле унутар процеса.

CASE-GENERATE клаузула је врло слична IF-GENERATE клаузули. Све што важи за IF-GENERATE важи и за CASE-GENERATE. Изглед CASE-GENERATE клаузуле се може видети у наредно блоку.

|  |
| --- |
| 01 **CASE** izraz **GENERATE**  02 **[**blok lokalnih deklaracija  03 **BEGIN]**  04 **WHEN** izbori **=>**  05 konkurentne klauzule  06 **WHEN** izbori **=>**  07 konkurentne klauzule  08 -- ...  09 **WHEN** **OTHERS** **=>**  10 konkurentne klauzule  11 **END** **GENERATE;** |

🛈 elsif, else и case су у generate клаузуле уведене у стандарду VHDL 2008, и постоји могућност да још нису подржане од стране свих алата за синтезу. For-generate и једноставан if-generate су синтетизабилни у свим алатима који су тренутно у употреби.

|  |
| --- |
| **Z** ПРИМЕР Померачки регистар са серијским улазом и паралелним излазом.  Uvedeno: if-generate; INOUT mod portova.  Пројектовати n-bitni (n - generic константа) померачки регистар са серијским улазом и паралелним излазом. На сваки такт се уводи по један бит на серијски улаз а n претходно уведених битова су присутни на паралелном излазу. Користити инстанце D flipflopa. |
| 01 **LIBRARY** ieee**;**  02 **USE** ieee**.**std\_logic\_1164**.ALL;**  03 ------------------------------------------  04 ------------------------------------------  05 ------------------------------------------  06 **ENTITY** shift\_reg **IS**  07 **GENERIC(**n**:** natural **:=** 4**);**  08 **PORT** **(**  09 clk**:** **IN** std\_logic**;**  10 serial\_data\_in**:** **IN** std\_logic**;**  11 parallel\_data**:** **INOUT** std\_logic\_vector**(**n**-**1 **DOWNTO** 0**)**  12 **);**  13 **END** **ENTITY** shift\_reg**;**  14 ------------------------------------------  15 ------------------------------------------  16 ------------------------------------------  17 **ARCHITECTURE** cell\_level **OF** shift\_reg **IS**  18 **BEGIN**  19  20 reg\_array**:** **FOR** index **IN** parallel\_data'**RANGE** **GENERATE**  21 **BEGIN**  22 first\_cell**:** **IF** index**=** parallel\_data'**left** **GENERATE**  23 -- prvi bit treba povezati na serijski ulaz  24 **BEGIN**  25 cell**:** **ENTITY** work**.**D\_flipflop**(**simple**)**  26 **PORT** **MAP** **(**clk**=>**clk**,**  27 d**=>**serial\_data\_in**,**  28 q**=>**parallel\_data**(**index**));**  29 **END** **GENERATE** first\_cell**;**  30 non\_first\_cell**:** **IF** index**/=** parallel\_data'**left** **GENERATE**  31 **BEGIN**  32 cell**:** **ENTITY** work**.**D\_flipflop**(**simple**)**  33 **PORT** **MAP** **(**clk**=>**clk**,**  34 d**=>**parallel\_data**(**index**+**1**),**  35 q**=>**parallel\_data**(**index**));**  36 **END** **GENERATE** non\_first\_cell**;**  37 **END** **GENERATE** reg\_array**;**  38 **END** **ARCHITECTURE** cell\_level**;** |
| У овом примеру искоришћен је елемент D FlipFlip који је направљен у претходном примеру.  Овде је IF-GENERATE употребљен у FOR-GENERATE клаузули. Прави се разлика између првог елемента и осталих, јер први елемент је повезан на улазни пин компоненте док су остали пинови повезани на излазе претходних елемената.    🛈 У овом примеру је искоришћен INOUT мод порта, да би порт могао да се чита и у њега уписује унутар архитектуре. Овим је избегнута декларација неког интерног сигнала који би прослеђивао податке са једног на други флипфлоп, и нема другу намену. Коришћење INOUT мода за ове намене није препоручљиво, из најмање два разлога. Интерфејс компоненте не би требало да буде диктиран интерном имплементацијом, већ имплементација треба да се прилагођава интерфејсу. Друго, INOUT портови захтевају пажљиву арбитражу у имплементацији, како би се избегли конфликти или изостанак побуде у неком тренутку. Из другог наведеног разлога, INOUT портове нећемо користити у овом курсу.  Да би се избегло коришћење INOUT порта у овом примеру, требало би декларисати сигнал у декларативном делу архитектуре; све Q и D портове флипфлопова треба повезати с тим сигналом. Изван generate клаузлуле тај интерни сигнал треба доделити излазном порту. |