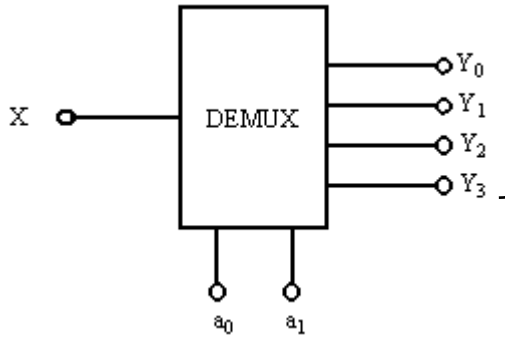


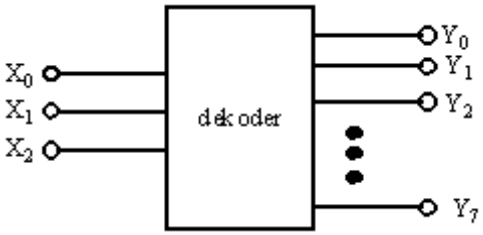
1. Demultipleksor – kombinacijski logički sklop
Demultipleksor je sklop koji prenosi ulaznu liniju na jednu izlaznu liniju na osnovi n-bitne adrese.

primjer demultipleksora sa dvobitnom adresom

2. Demux 2/4
ulazi: dva podatkovna (b, a), jedan adresni (g and c)
izlazi: četiri izlaza (y0, y1, y2, y3) – pri izlazu se invertiraju

3. Demux 3/8
ulazi: četiri podatkovna, dva adresna
izlazi: osam izlaza (y0, y1, y2, y3, y4, y5, y6, y7) – pri izlazu se invertiraju

4. Demultipleksor - sklop koji prenosi ulaznu liniju na jednu izlaznu liniju na osnovi n-bitne adrese
Dekoder - sklop koji ima onoliko ulaza koliko ima bitova kod i onoliko izlaza koliko se znakova može predočiti dotičnim kodom



- trobitni dekoder koji za svaki trobitni kod aktivira jednu od izlaznih linija

5. Ponašajni (funkcijski) model – sličan programiranju, tj. izradi programa – opisujemo ponašanje sklopa
Strukturni model – opisujemo sastavne elemente od kojih se sklop sastoji i način međusobnog povezivanja elemenata, a elementi koje tada koristimo su komponente

6. Varijabla - vrijednost im se ne može dodjeljivati uz definiranje kašnjenja
Signal – koristimo ih kao općenitu reprezentaciju vodiča, vrijednost im se može dodjeljivati uz definiranje kašnjenja

- razlika je u načinu uporabe

7. Blok PROCESS koji se ne smije pisati
Blok PROCESS se ne smije pisati bez da uvrstimo naredbu čekanja jer bi beskonačna petlja zaglavila rad simulatora. Nije preporučljivo niti pisati signale unutar ovoga bloka, jer ćemo dobiti krivi izlaz.

8. Dva ekvivalentna oblika bloka PROCESS:
Prvi oblik je oblik u kojemo prije završetka bloka navedemo naredbu WAIT ON (a, b) gdje će se blok zamrznuti dok ne dođe do promjene bilo varijable a, bilo varijable b.
Drugi oblik je taj gdje nakon riječi PROCESS u zagradu stavimo određene varijable, npr. (a, b), a blok će imat istu, tj. ekvivalentnu „funkciju“ kao i prvi oblik.

9. Oblici naredbe WAIT: WAIT ON (a, b), WAIT (trajno blokira izvođenje), WAIT FOR (možemo upisati vrijednost koliko će blok biti zamrznut)

10. Lista osjetljivosti – predstavlja signale u zagradi nakon riječi PROCESS, a blok je zamrznut dok se bilo koji signal ne promijeni, pa je jasno zašto postoji taj naziv (naravno, prvi put se blok izvrši, a dalje samo kada se barem jedan od signala ne promijeni).

11. Listu osjetljivosti: stavak_procesa ⇐ naziv: process (*lista osjetljivosti*)

12. WAIT ON (navedeni signali) naredba piše se prije kraja bloka PROCESS, a lista osjetljivosti na početku nakon riječi PROCESS gdje se u zagradi navode signali čije će promijena odmrznuti blok. Imaju istu „funkciju".

13. Multipleksor - na osnovi n-bitne adrese odabire jednu od 2^n ulaznih linija i prenosi je na izlaz, izborom ulazne linije upravlja skup varijabli odabira

14. Dekoder – identificira kodnu riječ nekog koda

15. Multipleksorsko stablo
a) podjela tablice definicija funkcija u podtablice – ulazi u MUX više razine
b) varijable viših težina na MUX „više težine"

16. Dekodersko stablo
- općenita metoda, vrijedi za proizvoljno složeni modul, npr izvedba dekodera sklopovima I

17. Komentari: u svakom redu dovoljno je započeti s - -

18. Kašnjenje u simulaciji ponašajnog modela
Ne postoji, takva simulaciji ispituje samo logički rad sklopa i pretpostavlja se da se sve odvija beskonačno brzo

19. Puno ime signala tijekom izvođenja simulacije
/uut/imesignala

20. Vrijeme kašnjenje sklopa saznajemo pri implementaciji skopa u ciljnoj tehnologiji, a on je jednak zbroju kašnjenja svakog pojedinog sklopa.

21. Multipleksor koji ima tri adresna ulaza ima 8 podatkovnih ulaza.

22. Višebitni signali
SIGNAL imesignala: std_logic_vector (donja_granica to gornja_granica)
ili
SIGNAL imesignala: std_logic_vector (gornja_granica downto donja_granica)

Kako se definiraju višebitni signali?

Deklaracija višebitnih signala slična je deklaraciji jednobitnih, ali se umjesto std_logic mora upotrijebiti std_logic_vector te se također mora definirati raspon.
U općem slučaju deklaracija višebitnih signala obavlja se na sljedeći način:
SIGNAL imesignala: std_logic_vector (donja_granica TO gornja_granica);
odnosno sa:
SIGNAL imesignala: std_logic_vector (gornja_granica DOWNTO donja_granica);

Jednom kada se odluči za način deklaracije (TO ili DOWNTO), taj način dalje treba poštivati pri radu s tim signalom. Umjesto riječi imesignala treba staviti naziv signala koji se deklarira. U slučaju da je istovremeno potrebno deklarirati više signala imena se razdvajaju zarezom.

Primjeri deklariranja višebitnih signala su:
-- Deklaracija internog 10-bitnog signala SIGNAL bus10: std_logic_vector (9 downto 0);
-- Deklaracija tri trobitna SIGNAL sigA, sigB, sigC: std_logic_vector (2 downto 0);
-- Deklaracija ulaznog 8-bitnog signala inbus: IN std_logic_vector (0 to 8);

Dodjeljivanje vrijednosti višebitnim signalima obavlja se upotrebom standardnog operatora za dodjeljivanje vrijednosti (<=), a konstantne – višebitne – vrijednosti se moraju upisivati unutar dvostrukih navodnika (").

U dodjeljivaju je također moguće korištenje raspona.
-- Inicijalizacija 3-bitnog signala kojemu je bit 2 bit najveće težine: sigA <= "011";
-- Prethodno dodjeljivanje je ekvivalentno sljedećem: sigA (2 downto 0) <= "011";
-- Dodjeljivanje 3-bitnog signala sigB signalu sigA širine 6-bita (sigA), pri čemu se ta 3 bita smještaju u bitove, 5, 4 i 3: sigA (5 downto 3) <= sigB;

U slučaju da je potrebno spajanje dva ili više signala u šire signale koristi se operator konkatenacije (&).
Primjer:
-- Spajanje dva 3-bitna signala (sigB i sigC) u jedan 6-bitni signal te dodjeljivanje te vrijednosti signalu sigA. sigB se smješta u više bitove signala sigA, a sigC u niže bitove signala sigA: sigA <= sigB & sigC;
Osim operatora konkatenacije za spajanje signala moguće je koristiti i zagradu, npr.:
-- Spajanje tri 3-bitna signala (sigB, sigC i sigD) u jedan 9-bitni signal te dodjeljivanje te vrijednosti signalu sigA. sigB se smješta u više bitove signala sigA, a sigD u niže bitove signala sigA. : sigA <= (sigB, sigC, sigD)

Ispravna uporaba CASE naredbe prilikom modeliranja kombinacijskih sklopova.

Kod IF naredbe potrebno je pripaziti da se prilikom modeliranja kombinacijskih sklopova uključe sve moguće kombinacije, jer u suprotnom se nakon sinteze može dobiti sekvencijski sklop, tj. sklop koji sadrži memorijske elemente. To konkretno znači da je prilikom pisanja modela kombinacijskog sklopa obavezno navesti ELSE izraz s odgovarajućim blokom naredbi, bez obzira što je taj dio u sintaksi naveden kao neobavezan.

Kod CASE naredbe je obavezno obuhvatiti sve moguće slučajeve koje izraz može poprimiti. To znači da se mora ili navesti potreban broj WHEN uvjeta (pa WHEN OTHERS ne pisati), ili navesti nedovoljan broj WHEN uvjeta i zatim kroz WHEN OTHERS pokriti preostale slučajeve. Evo primjera za razmisliti: definiran je dvobitni signal: SIGNAL sig: std_logic_vector(1 downto 0). Ukoliko želimo poduzeti odgovarajuću akciju za svaku moguću kombinaciju vrijednosti tog dvobitnog signala (dakle, pokriti je jednim WHEN uvjet), koliko ukupno WHEN uvjet izraza trebamo? (Hint: puno više od 4!) - Std_logic definira 9 vrijednosti za signale, mi ucimo samo 1,0,U

Način stvaranja primjeraka komponenti u strukturnom modeliranju:

Strukturni VHDL za razliku od ponašajnog opisuje od kojih jednostavnijih komponenti se sklop sastoji i na koji način su te komponente međusobno povezane. Opisi pojedinih komponenti mogu biti ponašajni, ali također mogu biti i strukturni.
Nakon ključne riječi ARCHITECTURE, a prije ključne riječi BEGIN, moraju se deklarirati i svi interni signali.

Interni signali služe za međusobno povezivanje komponenti. Kao pravilo može se uzeti da sve linije na nekoj shemi koje povezuju izlaze jedne komponente sa ulazom neke druge komponente treba deklarirati kao interni signal.

Komponente se upotrebljavaju u strukturnom opisu sklopa stvaranjem njihovih primjeraka (kažemo još instanciranjem). Stvaranje primjerka znači konkretnu upotrebu neke unaprijed deklarirane komponente. Prema tome, kada se govori o primjerku komponente govori se o konkretnom elementu u dizajnu, a kada se govori o komponenti misli se na sve elemente u dizajnu s istim sučeljem i ponašanjem. Stvaranje primjeraka i povezivanje komponenata obavlja se u bloku ARCHITECTURE između odgovarajućih rezerviranih riječi BEGIN i END.

Sintaksa za stvaranje primjerka je sljedeća:
ime_primjerka: ENTITY work.naziv_sklopa PORT MAP (povezivanje ulaza i izlaza);

Ako koristimo komponentu čije je sučelje:
ENTITY sklopI IS PORT (a, b: IN std_logic; z: OUT std_logic); END ENTITY;
Tada primjerak te komponente možemo stvoriti na sljedeći način:
i1: ENTITY work.sklopI PORT MAP (s1, s2, s3);
Primjerak je nazvan i1. Signal s1 povezuje se na a ulaz komponente, signal s2 na b ulaz komponente, te signal s3 na z izlaz komponente. U tom primjeru upotrebljen je implicitan način povezivanja, odnosno povezivanje putem pozicije. Osim implicitnog, moguće je i povezivanje putem imena:
i2: ENTITY work.sklopI PORT MAP (z => s3, b => s2, a => s1);
U ovom slučaju nije bilo nužno navoditi signale po redoslijedu kako su navođeni prilikom deklaracije komponente.

Tijekom spajanja komponenti ponekad je potrebno kombiniranje signala i dodavanje invertora. Pri tome je potrebno obratiti pažnju na nekoliko pravila koja treba poštivati. Unutar izraza PORT MAP na mjestu gdje se očekuje ulazni jednobitni signal, dopušteno je staviti konstantu ('0' ili '1'), signal 10 (s1) ili komplement signala (not s1). Nije dopušteno obavljati složene Booleove funkcije (and, or, ...). Time je dopušteno npr:
i1: ENTITY work.sklopI PORT MAP ('0', NOT s2, s3);
ali sljedeći izraz nije dopušten:
i2: ENTITY work.sklopI PORT MAP (NOT s1 OR s2, s2, s3);
Na mjesta na kojima se očekuje višebitni signal, može se dovesti odgovarajući višebitni interni signal, njegov dio ili pak agregacija signala, uz ograničenja koja će biti navedena u nastavku. Evo primjera: neka je definiran sklop sklopA čije je sučelje sljedeće:

```
ENTITY sklopA IS PORT (
ulaz1: std_logic_vector(3 downto 0);
ulaz2: std_logic_vector(0 to 1);
 izlaz: OUT std_logic);      END sklopA;
```

Prilikom stvaranja primjerka komponente, na mjestu višebitnih ulaznih signala mogu doći agregacije konstanti, na oba načina (koristeći zagrade ili operator &). Npr. ispravno je:

i3: ENTITY work.sklopA PORT MAP (('1','0','1','1'), '0'&'1', y);

Međutim, agregacija unutar PORT MAP izraza ne smije sadržavati signale (niti operacije sa signalima). Dakle, sljedeće je pogrešno (ako je a jednobitni signal):

i3: ENTITY work.sklopI PORT MAP (('1',a,'1','1'), '0'&'1', y);

Ovakav izraz može (a i ne mora) preživjeti prevođenje, međutim, pokretanje simulacije neće uspjeti. Da bismo riješili ovaj problem, mogu se definirati dva pomoćna interna signala (4-bitni int1 i 2-bitni int2), pa možemo pisati: int1 <= (a, not a, not a or b, '1');

int2 <= not a & (not b or c);

i3: ENTITY work.sklopI PORT MAP (int1, int2, y);

Alternativno ovom pristupu, možemo definirati jedan veliki interni signal (6-bitni interni), i zatim njegove dijelove koristiti za potrebne višebitne ulaze:

interni <= (a, not a, not a or b, '1', not a, not b or c);

i3: sklopI PORT MAP (interni(0 to 3), interni(4 to 5), y);

Agregacija signala.

Agregacija je jezicna konstrukcija iz VHDL-a koja se koristi za dodjelu vrijednosti objektu vektorskog tipa(signalima).

```
1  SIGNAL w: STD_LOGIC_VECTOR(7 DOWNTO 0);
2  . . .
3  w <= “00001001”;
4  w <= (’0’,’0’, ’0’, ’0’, ’1’, ’0’, ’0’, ’1’);
5  w <= (7=>’0’,1=>’0’,5=>’0’,6=>’0’,0=>’1’,4=>’0’,3=>’1’,2=>’0’);
```

U jeziku VHDL signali se definiraju ključnom riječi SIGNAL. Signale koristimo kao općenitu reprezentaciju vodiča; signalima se vrijednost može dodjeljivati uz definiranje kašnjenja. Primjerice, imamo li definiran signal s1, sljedeći izraz je legalan: s1 <= a and not b after 25 ns; i njime definiramo da će vrijednost tog signala uvijek odgovarati vrijednosti koju računa izraz a and not b, ali uz 25 nanosekundi kašnjenja; kad god se promijene bilo a, bilo b, signal s1 poprimit će vrijednost a and not b tek 25 nanosekundi nakon te promjene.

Signali se ključnom riječi SIGNAL definiraju u okviru definicije arhitekture sklopa, ali prije navođenja ključne riječi BEGIN.

```
ARCHITECTURE arch OF sklop IS
SIGNAL s1: std_logic;
SIGNAL s2: std_logic;
BEGIN  ....
```

Povezivanje po imenu / povezivanje po poziciji.

- Kod pozicijskog povezivanja signali se navode redosljedom kojim su deklarirani u sučelju korištene komponente
- Kod povezivanja putem imena redosljed nije bitan jer se navode izrazi oblika

„ime_porta_komponenta => naziv_signala“

Kako operatori I, ILI i NE djeluju nad argumentima čije vrijednosti nisu samo 0 i 1, već uključuju i vrijednost U (neinicijalizirano)? Primjerice, znamo da je NOT 1 = 0; a koliko iznosi NOT U? Prikažite djelovanja svih navedenih operatora tablično (također i za Ex-ILI)!

Tablica za **AND**

	0	1	U
0	0	0	0
1	0	1	U
U	0	U	U

Tablica za **OR**

	0	1	U
0	**0**	**1**	**U**
1	**1**	**1**	**1**
U	**U**	**1**	**U**

Tablica za **NOT**

0	1	U
1	0	U

Tablica za **EX-ILI**

	0	1	U
0	0	1	U
1	1	0	1
U	U	1	0

--ovo zadnje je po mojoj slobodnoj procjeni,ali mislim da je tocno

Prilikom strukturnog opisa nekog sklopa, taj sklop opisujemo koristeći komponente. Svaka od tih komponenti može biti opisana opet strukturno ili ponašajno. Ako se tako spuštamo do dna, kako je opisana ona najjednostavnija komponenta – strukturno ili ponašajno? Objasnite!

Nesto treba imati prvo strukturni opis da bi imalo ponasajni opis, barem tako mislim, u pripremi pisu gluposti. Sve komponenente mogu imati oba opisa. Kako god jednostavna struktura bila ona ima odredeno ponasanje :D