VHDL redosled izvršavanja naredbi

Kod klasičnih programskih jezika, kao što su C, Python ili Java, naredbe se izvršavaju jedna po jedna, redom kako su napisane.

```
a = 5;
b = a + 1;
```

Prvo se izvršava prva linija, pa tek onda druga.

VHDL redosled izvršavanja naredbi

Ali VHDL nije programski jezik za izvršavanje softvera, već jezik za opis hardvera.

A u hardveru se više stvari dešava ISTOVREMENO.

Na primer, u digitalnom kolu:

jedan AND gate radi,
jedan OR gate radi,
flip-flop radi,
brojač radi,

i svi oni rade paralelno u istom trenutku.

Zbog toga VHDL podržava:

konkurentno izvršavanje
sekvencijalno izvršavanje

VHDL redosled izvršavanja naredbi

KONKURENTNO IZVRŠAVANJE

Konkurentno znači:

sve naredbe rade paralelno.

To je prirodno ponašanje hardvera.

VHDL redosled izvršavanja naredbi

Primer 1 — konkurentne naredbe:

```
architecture primer of kolo is
begin

y1 <= a AND b;

y2 <= c OR d;

end architecture;
```

VHDL redosled izvršavanja naredbi

```
architecture primer of kolo is
begin

y1 <= a AND b;

y2 <= c OR d;

end architecture;
```

Ovde imamo dve potpuno nezavisne logičke operacije:

Prva: y1 <= a AND b;

predstavlja jedno AND kolo.

Druga:

```
y2 <= c OR d;
predstavlja jedno OR kolo.
U FPGA-u ili digitalnom kolu ova dva kola fizički postoje istovremeno.
Zato se obe naredbe izvršavaju paralelno.
```

Ovde redosled pisanja NIJE važan.

Možemo zameniti mesta naredbama:

```
y2 <= c OR d;
y1 <= a AND b;
i hardver će biti potpuno isti.
```

Konkurentna i sekvencijalna dodela signala

1. Konkurentna dodela

Kod konkurentne dodele:

sve naredbe rade paralelno,
redosled pisanja uglavnom nije važan,
koristi se direktno u arhitekturi.

To najviše liči na stvarni hardver.

Konkurentna i sekvencijalna dodela signala

1. Konkurentna dodela

Kod konkurentne dodele:

sve naredbe rade paralelno,
redosled pisanja uglavnom nije važan,
koristi se direktno u arhitekturi.

To najviše liči na stvarni hardver.

```
architecture primer of kolo is
begin

    y1 <= a AND b;

    y2 <= c OR d;

    y3 <= a XOR d;

end architecture;
```

Konkurentna i sekvencijalna dodela signala

```
architecture primer of kolo is
begin

    y1 <= a AND b;

    y2 <= c OR d;

    y3 <= a XOR d;

end architecture;
```

Ovde FPGA pravi TRI logička kola:

jedno AND kolo,

jedno OR kolo,

jedno XOR kolo.

I sva tri rade ISTOVREMENO.

Nijedno ne čeka drugo.

To znači:

ako se promeni signal a,

odmah se ažuriraju i y1 i y3.

Konkurentna i sekvencijalna dodela signala

2. Sekvencijalna dodela

Sekvencijalna dodela znači:

naredbe se izvršavaju redom.

Koristi se unutar PROCESS bloka.

Ovde redosled postaje važan.

```
process(a, b, c)
begin

    x <= a AND b;

    y <= x OR c;

end process;
```

Ovde imamo proces.

Proces predstavlja deo hardvera koji izvršava naredbe redom.

Prvo se računa:

```
x <= a AND b;
a zatim:
y <= x OR c;
```

Concurrent assignment (konkurentna dodela)

Do sada smo koristili operator:

<=

Ovaj operator služi za dodelu vrednosti signalima.

Ali u VHDL-u to nije obična programska dodela kao u C-u.

Concurrent assignment (konkurentna dodela)

U VHDL-u:

y <= a AND b;

ne znači:

„izračunaj pa sačuvaj u memoriju“

nego:

„napravi hardversku vezu između izlaza i logike.“

Concurrent assignment (konkurentna dodela)

U VHDL-u:

y <= a AND b;

ne znači:

„izračunaj pa sačuvaj u memoriju“

nego:

„napravi hardversku vezu između izlaza i logike.“

Concurrent assignment znači:

naredbe rade paralelno,

više signala može da se ažurira istovremeno,

opisujemo realni hardver.

Concurrent assignment (konkurentna dodela)

```
architecture primer of kolo is
begin

    y1 <= a AND b;

    y2 <= a OR c;

    y3 <= y1 XOR y2;

end architecture;
```

Ovde FPGA pravi tri logička kola:

AND

OR

XOR

Sva tri postoje istovremeno u hardveru.

Ako se promeni a:

AND kolo odmah računa novi y1

OR kolo odmah računa novi y2

XOR kolo odmah računa novi y3

Sve se dešava skoro simultano.

Concurrent assignment (konkurentna dodela) with i when

Pored obične konkurentne dodele:

```
y <= a AND b;
```

VHDL podržava i naprednije konkurentne naredbe:

- with select
- when else

One služe za:

multipleksere,
dekodere,
logiku izbora,
uslovno generisanje izlaza.

Concurrent assignment (konkurentna dodela) with i when

```
y <= a when sel = '0' else b;
```

Ako je sel=0 izlaz je a.

Ako je sel=1 izlaz je b.

Process i konkurentne naredbe zajedno

```
architecture proba of Opsti_primer is
begin

    c2 <= ad OR ale;

    c1(0) <= rd AND wr(0);

    seq: process(wr(7))
    begin

    end process seq;

end architecture proba;
```

Process i konkurentne naredbe zajedno

```
architecture proba of Opsti_primer is
begin

    c2 <= ad OR ale;

    c1(0) <= rd AND wr(0);

    seq: process(wr(7))
    begin

    end process seq;

end architecture proba;
```

Linija 1

```
c2 <= ad OR ale;
Ovo opisuje jedno OR kolo.
Kad god se promeni:
ad
ili
ale
izlaz c2 se odmah ažurira.
```

Linija 2

```
c1(0) <= rd AND wr(0);
Ovo opisuje jedno AND kolo.
Dakle sada već imamo DVA paralelna kola:
OR
AND
```

Linija 3

```
seq: process(wr(7))
Ovde počinje PROCESS.
```

Process i konkurentne naredbe zajedno

```
architecture proba of Opsti_primer is
begin

    c2 <= ad OR ale;

    c1(0) <= rd AND wr(0);

    seq: process(wr(7))
    begin

    end process seq;

end architecture proba;
```

Ime:

`seq:`
`je labela procesa.`
`Nije obavezna, ali je veoma korisna.`
`Kasnije u simulaciji ili sintezi možemo lakše pronaći taj proces.`

Process i konkurentne naredbe zajedno

```
architecture proba of Opsti_primer is
begin

    c2 <= ad OR ale;

    c1(0) <= rd AND wr(0);

    seq: process(wr(7))
    begin

    end process seq;

end architecture proba;
```

UNUTAR procesa:

naredbe su sekvencijalne.

ALI:

sam process radi konkurentno sa ostatkom arhitekture.

Process i konkurentne naredbe zajedno

```
architecture proba of Opsti_primer is
begin

    c2 <= ad OR ale;

    c1(0) <= rd AND wr(0);

    seq: process(wr(7))
    begin

    end process seq;

end architecture proba;
```

Ime:

```
seq:
je labela procesa.
Nije obavezna, ali je veoma korisna.
Kasnije u simulaciji ili sintezi možemo lakše pronaći taj proces.
```

process(wr(7))

To je sensitivity list.

To znači:

proces će se pokrenuti SAMO kada se promeni signal wr(7).

Process i konkurentne naredbe zajedno

```
architecture proba of Opsti_primer is
begin

    c2 <= ad OR ale;

    c1(0) <= rd AND wr(0);

    seq: process(wr(7))
    begin

    end process seq;

end architecture proba;
```

UNUTAR procesa:

naredbe su sekvencijalne.

ALI:

sam process radi konkurentno sa ostatkom arhitekture.

Architecture = više paralelnih celina

Inside process = sequential execution

Selekciono dodeljivanje (with select)

```
with signal_select select
    izlaz <= vrednost1 when stanje1,
             vrednost2 when stanje2,
             vrednost3 when stanje3;
```

with select predstavlja:

hardversku logiku izbora.

To nije IF naredba iz softvera.

To je kombinaciono kolo.

Selekciono dodeljivanje (with select)

```
with sel select
    y <= a when "00",
         b when "01",
         c when "10",
         d when "11";
```

Linija 1

```
with sel select
Signal sel određuje šta će biti povezano na izlaz.
On je selektor multipleksera.
```

Linija 2

```
y <= a when "00",
Ako je:
sel = 00
onda izlaz postaje:
y = a
```

Linija 3

```
b when "01"
Ako je:
sel = 01
onda:
y = b
```

Isto za ostale slučajeve

```
10 -> c
11 -> d
```

Uslovno dodeljivanje (when else)

```
izlaz <= vrednost1 when uslov else
         vrednost2;
```

```
y <= a when sel = '0' else
     b;
```

Deo 1

```
sel = '0'
Ovo je uslov.
```

Deo 2

```
y <= a
Ako je uslov tačan:
sel = 0
izlaz postaje:
y = a
```

Deo 3

```
else b
Ako uslov NIJE tačan:
sel = 1
onda:
y = b
```

Uslovno dodeljivanje (when else)

```
izlaz <= vrednost1 when uslov else
         vrednost2;
```

```
y <= a when sel = '0' else
     b;
```

Hardver već postoji.

MUX samo bira koji signal prolazi na izlaz.

Deo 1

```
sel = '0'
Ovo je uslov.
```

Deo 2

```
y <= a
Ako je uslov tačan:
sel = 0
izlaz postaje:
y = a
```

Deo 3

```
else b
Ako uslov NIJE tačan:
sel = 1
onda:
y = b
```

Uslovno dodeljivanje (when else)

with select

Koristi se kada imamo:

više stanja,

više opcija,

jasno definisane kombinacije.

when else

Koristi se kada imamo:

jednostavan uslov,

IF-ELSE logiku,

jednostavan izbor.

LED kontrola

```
led <= '1' when button = '1' else
       '0';
```

Ako je taster pritisnut:

```
button = 1
onda LED svetli.
Inače je
iskljucena.
```

```
y <= a when enable = '1' else
     '0';
```

Kada je enable=1:

signal a prolazi na izlaz.

Kada je enable=0:

izlaz je nula.

To je praktično:

enable logika.

PROCESS u VHDL-u

PROCESS predstavlja:

sekvencijalni deo VHDL-a.

Van procesa

Naredbe su:

konkurentne

paralelne

Unutar procesa

Naredbe su:

sekvencijalne

izvršavaju se redom

Dobra analogija

„Architecture je kao fabrika sa više mašina koje rade paralelno.“

„Process je jedna mašina koja izvršava korake redom.“

PROCESS u VHDL-u

Osnovna struktura procesa

```
process(lista_signala)
begin

    naredbe;

end process;
```

1. process

Označava početak procesa.

2. lista signala

```
process(a, b, c)
To je sensitivity list.
Proces će se pokrenuti kada se promeni:
a
ili b
ili c
```

3. begin

Počinje telo procesa.

4. naredbe

Ovde se naredbe izvršavaju:

redom.

PROCESS u VHDL-u

```
process(a, b)
begin

    x <= a AND b;

    y <= a OR b;

end process;
```

Kada se proces pokreće?

Kada se promeni:

a

ili

b

Šta se prvo izvršava?

```
x <= a AND b;
```

Šta se izvršava posle?

```
y <= a OR b;
```

Veoma važna stvar

Unutar procesa:

redosled naredbi JE važan.

To je ogromna razlika u odnosu na architecture deo.

PROCESS u VHDL-u

```
process(clk)
begin

    if rising_edge(clk) then
        q <= d;
    end if;

end process;
```

Ovo opisuje:

D flip-flop.

Kada dođe rastuća ivica clock-a:

vrednost d

upisuje se u q.

PROCESS u VHDL-u

```
process(clk)
begin

    if rising_edge(clk) then
        q <= d;
    end if;

end process;
```

Ovo opisuje:

D flip-flop.

Kada dođe rastuća ivica clock-a:

vrednost d

upisuje se u q.