Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Digitalni Elektronski Sistemi

Osnove jezika VHDL

opis obnašanja vezja

Postopkovni opis vezja

Nivoji opisa vezja

- postopkovni (behavioral)
- funkcijski (dataflow, RTL)

```
stev: process (clk)
begin
if rising_edge(clk) then
   if reset='1' then
      q <= "00000000"
   else
      q <= q + 1;
   end if;
end if;
end process;
```

npr. opišemo delovanje števca

- ob resetu gre izhod na 0
- sicer pa se izhod povečuje za 1

Drugi visokonivojski jeziki za opis vezij

Verilog

```
module stevec (clk, reset, st);

input clk;
input reset;
output [7:0] st;
reg [7:0] st;

always @(posedge clk) begin: P
    if (reset) begin
        st <= 0;
    end
    else begin
        st <= (st + 1);
    end
end

endmodule
```

MyHDL (Python)

```
from myhdl import *

def stevec(clk, reset, st):

    @always(clk.posedge)
    def p():
        if reset:
            st.next = 0
        else:
            st.next = st + 1
    return p

clk = Signal(bool(0))
reset = Signal(bool(0))
st = Signal(intbv(0)[8:])
```

Primerjava VHDL – Verilog

- **Prednosti jezika VHDL (IEEE 1076-2002, 2008)**
  - zaradi strožjih sintaktičnih pravil dovoli manj napak
  - veliko podatkovnih tipov, naštevni tip (stanja), paketi
  - enostavnejša inicializacija pomnilnika
  - ni zmede glede vrste signalov: reg / wire (Verilog -)

- **Prednosti jezika Verilog (IEEE 1364-2005)**
  - sintaksa podobna jeziku C
  - kompaktna koda, komentarji
  - ne potrebuje veliko kode za komponente (VHDL -)

Procesno okolje

- Znotraj procesnega okolja so sekvenčni stavki
  - vrstni red stavkov je v procesu lahko pomemben
  - med sekvenčne stavke spada pogojni (if) stavek
  - dovoljeno je narediti več prireditev enemu signalu, dejansko se izvrši le zadnja prireditev

```
primerjava: process (a, b)
begin
   enako <= '0';
   if a=b then
      enako <= '1';
   end if;
end process;
```

Zgradba procesnega okolja

```
oznaka: process (seznam signalov)
begin
.....
end process;
```

- Z oznako poimenujemo del vezja, ki ga predstavlja proces
- Simulacija procesa se izvrši ob spremembi enega izmed signalov iz seznama
  - pri opisu kombinacijskih vezij moramo navesti vse signale, ki predstavljajo vhode v proces

Pogojni stavek (if)

- Ob izpolnjenem pogoju se izvrši en ali več stavkov, ki so zapisani za "then"
- Opcija: če pogoj ni izpolnjen se izvršijo stavki za "else"
- Pogojni stavek zaključimo z "end if "

```
if pogoj then
   stavki(1);
else
   stavki(2);
end if;
```

```
if pogoj1 then
   stavki(1);
elsif pogoj2 then
   stavki(2);
else
   stavki(3);
end if;
```

Primerjava s sočasnimi stavki

- Pogojni stavek spada med sekvenčne stavke
  - uporabljamo ga lahko le znotraj procesa
- Pogojni prireditveni stavek je alterativa med sočasnimi stavki

```
p_max: process(a, b)
begin
  if a>b then
     max <= a;
  else
     max <= b;
  end if;
end process;
```

=

```
max <= a when a>b else b;
```

Primerjava (2)

```
bin2bcd: process(bin)
begin
  if bin>9 then
     enice <= bin – “1010”;
     desetice <= ‘1’;
  else
     enice <= bin;
     desetice <= ‘0’;
  end if;
end process;
```

=

```
enice <= bin - ”1010”  when bin>9
                 else bin;

desetice <= ‘1’ when bin>9
                      else ‘0’;
```

- V pogojnem stavku imamo lahko več prireditev

Zaporedje pogojev: prioriteta

```
p: process(int0, int1, int2)
begin
  if int0='1' then
    v <= "01"
  elsif int1='1' then
    v <= "10";
  elsif int2='1' then
    v <= "11";
  else
    v <= "00";
  end if;
end process;
```

=

```
v <="01"  when int0='1' else
      "10"  when int1='1' else
      "11"  when int2='1' else
      "00";
```

int0 ima prednost pred
int1, ki ima prednost pred
int2 ...

- Zaporedni pogoji se ovrednotijo s prioriteto

Sekvenčni izbirni stavek (case)

- Odločamo se glede na vrednost enega signala
- Izbirni stavek nima prioritete

```
case ime_signala is
    when vrednost1 =>
       stavki(1);
    when vrednost2 =>
       stavki(2);
    ...
    when others  =>
       stavki(n);
  end case;
```

```
decod: process(digit)
begin
  case digit is
    when “00” =>
       display <= “00111111”;
    when “01” =>
       display <= “00000110”;
    when others  =>
       display <= “11111001”;
  end case;
end process;
```

Primer: izbiralnik (multipleksor)

- Izbiralnik s štirimi vhodi (a, b, c in d)

```
m: process(mode,a,b,c,d)
begin
  if mode="00" then
     mux <= a;
  elsif mode="01" then
     mux <= b;
  elsif mode="10" then
     mux <= c;
  else
     mux <= d;
  end if;
end process;
```

```
m: process(mode,a,b,c,d)
begin
  case mode is
    when "00" =>
       mux <= a;
    when "01" =>
       mux <= b;
    when "10" =>
       mux <= c;
    when others  =>
       mux <= d;
  end case;
end process;
```

Sočasni izbirni stavek (with…select)

- Na podlagi izbire priredimo signalu različne vrednosti (ali izraze)

```
with izbira select
signal <= izraz(1) when vrednost1,
          izraz(2) when vrednost1,
          ...
          izraz(n) when others;
```

- Za razliko od case stavka lahko prirejamo vrednost le enemu signalu

```
with digit select
display <= “00111111” when “00”,
           “00000110”  when “01”,
           “11111001”  when others;
```

Sekvenčni stavki in sinteza vezja

- Prirejanje vrednosti signalom na več mestih
- Če signalu pod kakšnim pogojem ne določimo vrednosti, se bo ohranjala zadnja vrednost
  - dobimo sekvenčno vezje !

```
p: process(set_flag, clear_flag)
begin
  if set_flag='1' then
     flag <= '1';
  elsif clear_flag='1' then
     flag <= '0';
  end if;
end process;
```

Naredili smo asinhroni zapah za signal flag !

Opis kombinacijskih vezij

- Definiramo vrednosti pri vseh pogojih:
  - vrednost priredimo pri vsakem "if" in "else" ali
  - vrednost priredimo na začetku procesa in jo spremenimo v "if" stavkih

```
primerjava: process (a, b)
begin
   if a=b then
      enako <= '1';
   else
      enako <= '0';
   end if;
end process;
```

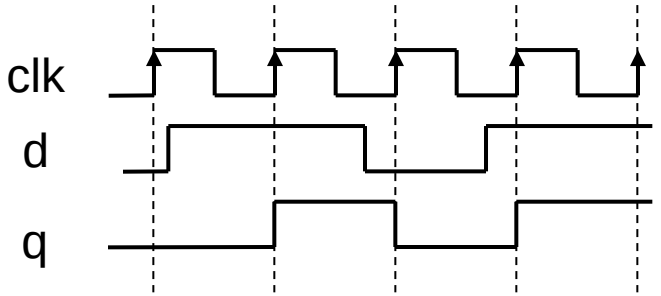
```
primerjava: process (a, b)
begin
   enako <= '0';
   if a=b then
      enako <= '1';
   end if;
end process;
```

Opis sekvenčnih vezij

- Izhodi sinhronih vezij se spreminjajo ob uri
  - prva fronta: clk'event and clk='1' ali rising_edge(clk)
  - zadnja: clk'event and clk='0' ali falling_edge(clk)

```
FF: process (clk)
begin
   if rising_edge(clk) then
      q <= d;
   end if;
end process;
```

Podatkovni flip-flop



Sinteza vezja s flip-flopi

- Narediti želimo sinhrona vezja s flip-flopi (FF)
  - ne želimo asinhronih pomnilnih elementov (latch)
  - za sintezo FF uporabimo ustrezno obliko zapisa

I. oblika: popolnoma sinhrono vezje

```
FF: process (clk)
begin
   if rising_edge(clk) then
      q <= d;
   end if;
end process;
```

II. oblika: asinhroni reset

```
FF: process (clk, reset)
begin
   if reset='1' then
      q <= "00000000";
   elsif rising_edge(clk) then
      q <= d;
   end if;
end process;
```

Povratne zanke

- Sinhrono vezje ima lahko povratno zanko
  - na podlagi stanja izhoda izračunamo novo stanje
  - Npr. števec, pomikalni register, sinhroni avtomat

I. oblika

```
stev: process (clk)
begin
   if rising_edge(clk) then
      q <= q + 1;
   end if;
end process;
```

- na podlagi stanja izhoda izračunamo novo stanje
- če je signal **q** zunanji, mora biti vrste **buffer**

Primer: BCD števec

- BCD števec šteje od 0 do 9

I. oblika

```
stev: process (clk)
begin
   if rising_edge(clk) then
      if q < 9 then
         q <= q + 1;
      else
         q <= “0000”;
      end if;
   end if;
end process;
```

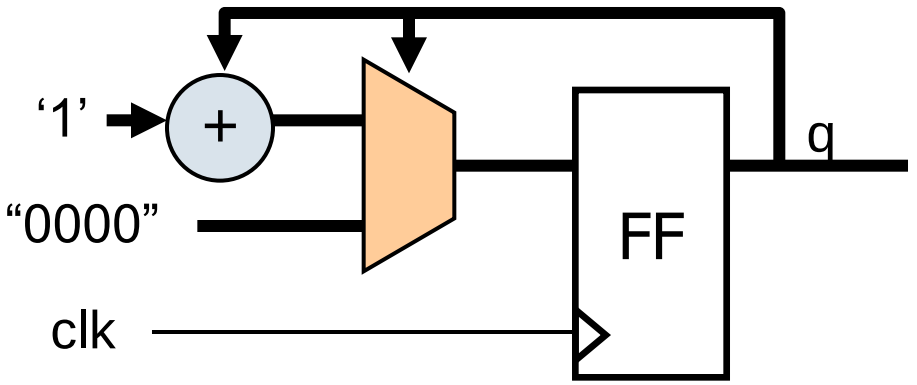
II. oblika

```
stev: process (clk)
begin
 if reset=‘1’ then
       q <= “0000”
 elsif rising_edge(clk) then
    if q < 9 then
       q <= q + 1;
    else
       q <= “0000”;
    end if;
 end if;
end process;
```

Zgradba vezja BCD števca

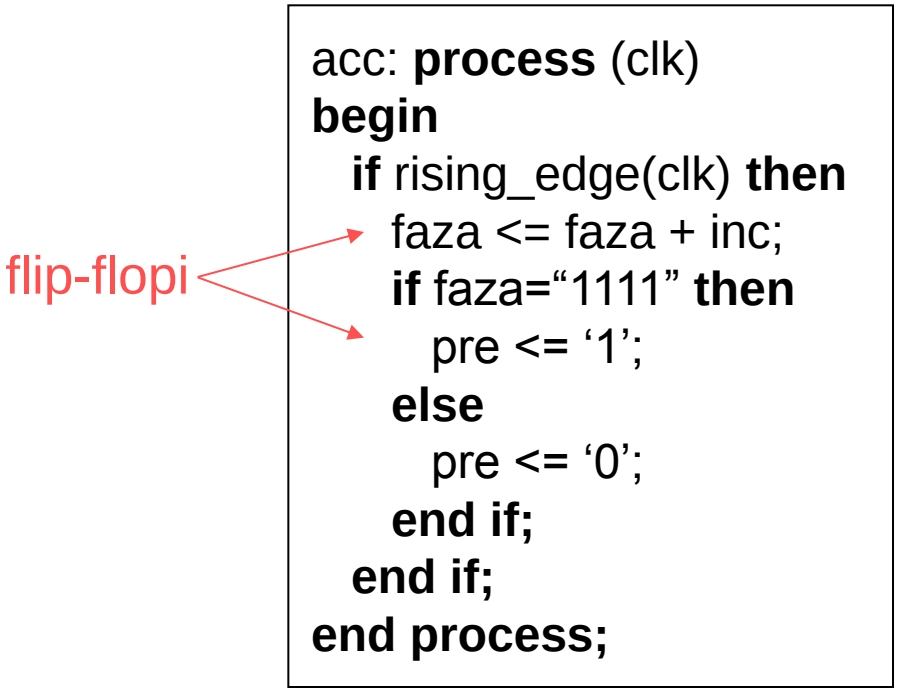
- Signal q je izhod FF, ki imajo na vhodu logiko
  - podatkovni vhod je odvisen od trenutnega izhoda – povratna zanka

```
stev: process (clk)
begin
   if rising_edge(clk) then
      if q < 9 then
         q <= q + 1;
      else
         q <= “0000”;
      end if;
   end if;
end process;
```



Kdaj dobimo flip-flope ?

- Kadar priredimo vrednost signalu ob fronti (ure), bo ta signal izhod iz flip-flopa(ov)



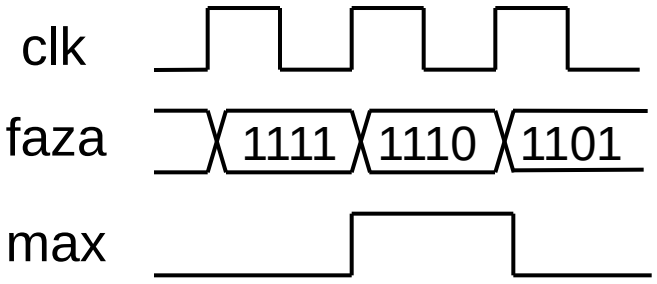
Potek simulacije

- Procesni stavek se izvede ob spremembi ure (clk)
  - pogoj rising_edge() je izpolnjen, ko gre clk iz 0 na 1

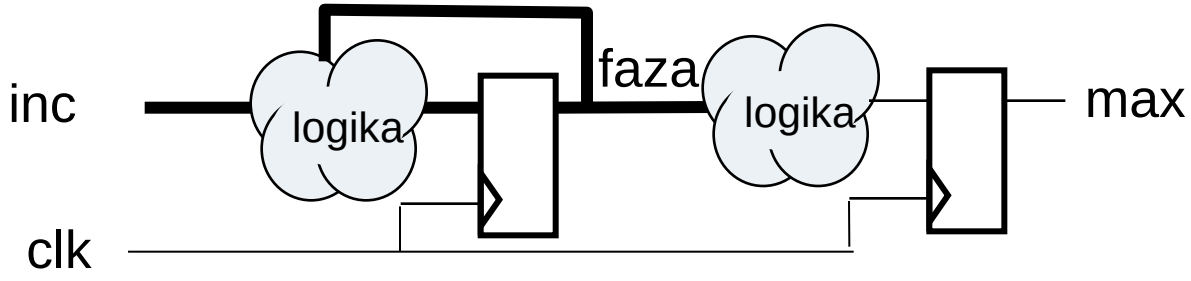
```
acc: process (clk)
begin
   if rising_edge(clk) then
      faza <= faza + inc;
      if faza="1111" then
         max <= '1';
      else
         max <= '0';
      end if;
   end if;
end process;
```

korak	**čas**	clk	inc	faza	max
čakaj	0	0, 1	1111	0000	0
račun	T	1	1111	1111	0
izvrši	T+d	1	1111	1111	0
čakaj	T2	0, 1	1111	1111	0
račun	T2+d	1	1111	1110	1
izvrši	T2+d	1	1111	1110	1

Časovni diagram in zgradba vezja



- Signal max se postavi na '1' šele ob naslednji fronti ure
  - zakasnitev, ker je izhod iz flip-flopa !



Pravila za modeliranje sekvenčnih vezij

- Sekvenčni del z izhodnimi flip-flopi opišemo s sinhronim procesom
- Kombinacijsko vezje opišemo s kombinacijskim procesom ali izven procesnega okolja

Register

Transfer

Level

```
acc: process (clk)
begin
   if rising_edge(clk) then
      faza <= faza + inc;
   end if;
end process;

max <= '1' when faza="1111"
           else '0';
```