Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Digitalni Elektronski Sistemi

Načrtovanje v jeziku VHDL

Pravila načrtovanja v jeziku VHDL

Pravila

1. Vezje naj bo sinhrono
   - ne uporabljaj asinhronih signalov
2. Pravilno uporabljaj seznam signalov v procesu
   - vsi vhodi v kombinacijsko vezje
   - signal clk za sekvenčno vezje
   - le v testni strukturi je seznam lahko prazen
3. Določi vrednosti vsem izhodom,
   - pri kombinacijskem vezju v vseh primerih
4. Priredi vrednosti signalu le v enem procesu
5. Ne testiraj vrednosti X ali Z
6. Ne določaj zakasnitev v opisu vezja, ki ga boš sintetiziral
7. Ne delaj kombinacijskih zank

1. Vezje naj bo sinhrono

- V vezju naj bo ena (globalna) ura
  - namesto deljenih signalov za uro raje uporabi hitro uro in pogoj za omogočanje (Clock Enable)
- Flip-flopi določajo signale za vhode komb. logike, ki ima izhode vezane na flip-flope
  - v vsakem urnem ciklu je le ena sprememba vrednosti signala
- Če je le možno NE uporabljaj asinhronih reset signalov
  - asinhroni signali morajo biti brez motenj
- Ne uporabljaj nivojsko proženih zapahov (latch)
  - v vezju naj bodo le robno proženi flip-flopi

1. Vezje naj bo sinhrono

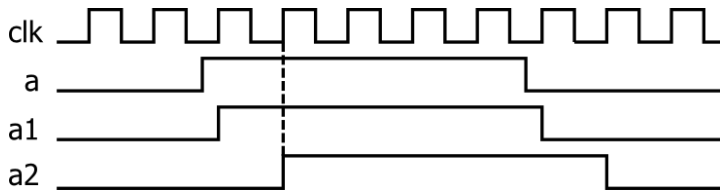
- Uporabi le en pogoj za fronto ure
  - sicer se vezje ne bo dalo sintetizirati!

```
p: process(start, stop)
begin
   if rising_edge(start) then
    en_reg <= '1';
   elsif rising_edge(stop) then
    en_reg <= '0';
   end if;
end process;
```

```
p1: process(start, stop)
begin
   if stop='1' then
     en_reg <= '0';
   elsif rising_edge(start) then
     en_reg <= '1';
   end if;
end process;
```

1. Vezje naj bo sinhrono

- sinhrono zaznavanje fronte
  - potrebujemo hitro uro (clk)
  - najboljša rešitev !




```
p: process (clk)
begin
   if rising_edge(clk) then
      start1 <= start; start2 <= start1
      stop1 <= stop;  stop2 <= stop1;
      if start1='1' and start2='0' then
         en_reg <= '1';
      elsif stop1='1' and stop2='0' then
         en_reg <= '0';
      end if;
   end if;
end process;
```

1. Vezje naj bo sinhrono

- izogibaj se asinhronim signalom
  - OK, če imamo signal reset, ki je brez motej in sinhroniziran

```
stev1: process (clk, reset)
begin
if reset='1' then -- asinhroni
   q <= "00000000";
elsif rising_edge(clk) then
   q <= q + 1;
end if;
end process;
```

- boljša rešitev

```
stev2: process (clk)
begin
if rising_edge(clk) then
   if reset='1' then
      q <= "00000000";
   else
      q <= q + 1;
   end if;
end if;
end process;
```

2. Pravilno določi seznam signalov v procesu

- v kombinacijskem vezju morajo biti vsi vhodi

```
p: process (state)
begin
  if reset='1' then
       next <= HG;
       start <= '1';
   else
       case state is
         when HG =>
              if cars='1' and long='1' then
                 next <= HY;
              else
                 next <= HG;
...
```

```
p: process (state, reset, cars, long)
begin
  if reset='1' then
       next <= HG;
       start <= '1';
   else
       case state is
         when HG =>
              if cars='1' and long='1' then
                 next <= HY;
              else
                 next <= HG;
...
```

2. Seznam signalov v sekvenčnem procesu

- v seznamu je le ura in morebitni asinhroni reset

```
process (Clk, D)
begin
  if rising_edge(Clk) then
    Q <= D;
  end if;
end process;

process (Clk, D)
begin
  if reset = '1' then
    Q <= '0';
  else
    if rising_edge(Clk) then
      Q <= D;
    end if;
  end if;
end process;
```

```
process (Clk)
begin
  if rising_edge(Clk) then
    Q <= D;
  end if;
end process;

process (Clk, reset)
begin
  if reset = '1' then
    Q <= '0';
  else
    if rising_edge(Clk) then
      Q <= D;
    end if;
  end if;
end process;
```

3. Določi vrednosti izhodom v vseh primerih

- sicer bodo v vezju zapahi

```
p: process (state, input)
begin
   case state is
      when S1 =>
         if input='1' then
            output <= '0';
         end if;

      when S2 =>
         output <= '1';
   end case;
end process;
```

```
p: process (state, input)
begin
   case state is
      when S1 =>
         if input='1' then
            output <= '0';
         else
            output <= '1';
         end if;

      when S2 =>
         output <= '1';
   end case;
end process;
```

3. Določi vrednosti izhodom v vseh primerih

```
dovoljen_prehod <= '1' when luc=zelena;
```

```
d: process(luc)
begin
 if luc=zelena then
   dovoljen_prehod <= '1';
 elsif luc=rdeca then
  dovoljen_prehod <= '0';
 end if;
end process;
```

3. Določi vrednosti izhodom v vseh primerih

- določi privzete vrednosti

```
-- dobro

p: process (state, input)
begin
    case state is
      when S1 =>
        if input='1' then
            output <= '0';
        else
            output <= '1';
        end if;

      when S2 =>
           output <= '1';
    end case;
end process;
```

```
-- bojše

p: process (state, input)
begin
    output <= '1';
    case state is
      when S1 =>
        if input='1' then
            output <= '0';
        end if;

      when S2 =>
           output <= '1';
    end case;
end process;
```

4. Vrednost priredimo le v enem procesu!

```
p1: process(clk)
begin
 if rising_edge(clk) then
  if tipka='1' then
    delam_reg <= '1';
  end if;
 end if;
end process;
```

```
p2: process(clk)
begin
 if rising_edge(clk) then
    q <= q + 1;
    if q=5 then
      delam_reg <= '0';
    end if;
  end if;
end process;
```

```
p1: process(clk)
begin
 if rising_edge(clk) then
  if tipka='1' then
    delam_reg <= '1';
  elsif q=5 then
    delam_reg <= '0';
  end if;
 end if;
end process;
```

```
p2: process(clk)
begin
  if rising_edge(clk) then
     q <= q + 1;
  end if;
end process;
```

5. Ne primerjaj vrednosti z X ali Z!

- Deluje na simulaciji, vendar ne po sintezi!

```
architecture behv of ALU is begin
  process (A,B,Sel) begin
    case Sel is
      when "00" => Res <= A + B;
      when "01" => Res <= A + (not B) + 1;
      when "1X" => Res <= A and B;
      when "1Z" => Res <= A or B;
      when others => Res <= "XX";
    end case;
  end process;
end behv;
```

```
architecture behv of ALU is begin
  process(A,B,Sel) begin
    case Sel is
      when "00" => Res <= A + B;
      when "01" => Res <= A + (not B) + 1;
      when "10" => Res <= A and B;
      when "11" => Res <= A or B;
      when others => Res <= "XX";
    end case;
  end process;
end behv;
```

6. Ne uporabljaj zakasnitev v opisu vezja

- Zakasnitve so del simulacijske testne strukture
  - programu za sintezo ne moremo določiti zakasnitev

    a <= c **xor** d **after** 20 **ns**;
- Ne uporabljaj procesov s stavki wait za sintezo vezja !

7. Ne delaj kombinacijskih zank

- Povratna vezava v kombinacijskem vezju ni dobra praksa

  C <= **not** C;

  a <= a + 1;

  - vezje bo osciliralo ali delovalo le v posebnih primerih

Preglej izsek VHDL kode in ugotovi

- Kateri signali so izhodi ?
- V izseku VHDL kode smo prekršili naslednje pravilo:
- Kateri so kombinacijski in kateri sekvenčni deli vezja?

```
p_rgb: process (blank,x,y,color)
begin
 if blank='0' then
  if x>=256 and x<512 and y>=256 and y<272 then
    rgb <= color;
  else
    rgb <= "011";
  end if;
 else
   rgb <= "000";
 end if;
end process;

rgb <= "000";
```

4!

Preglej izsek VHDL

- Kateri signali so izhodi ?
- V izseku VHDL kode smo prekršili naslednje pravilo:
- Kateri so kombinacijski in kateri sekvenčni deli vezja?

2!

1!

```
p2: process(extclk, clk)
begin
 if rising_edge(clk) then
   delclk <= extclk;
 end if;

 datamask <= data and maska;
end process;
```

Preglej izsek VHDL

- Kateri signali so izhodi ?
- V izseku VHDL kode smo prekršili naslednje pravilo:
- Kateri so kombinacijski in kateri sekvenčni deli vezja?

2!

3!

```
p: process(x, y, sel)
begin
 case sel is
  when "00" =>
    xi <= x;
    yi <= y;
  when "01" =>
    xi <= y;
    yi <= STD_LOGIC_VECTOR(Tx);
end case;
end process;
```