Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

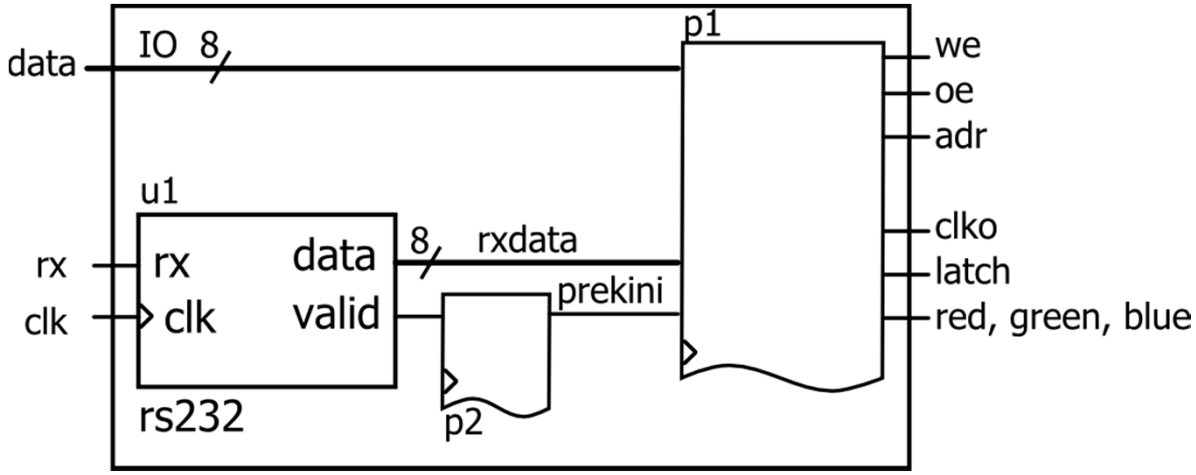
Digitalni Elektronski Sistemi

Načrtovanje v jeziku VHDL

Pravila načrtovanja v jeziku VHDL

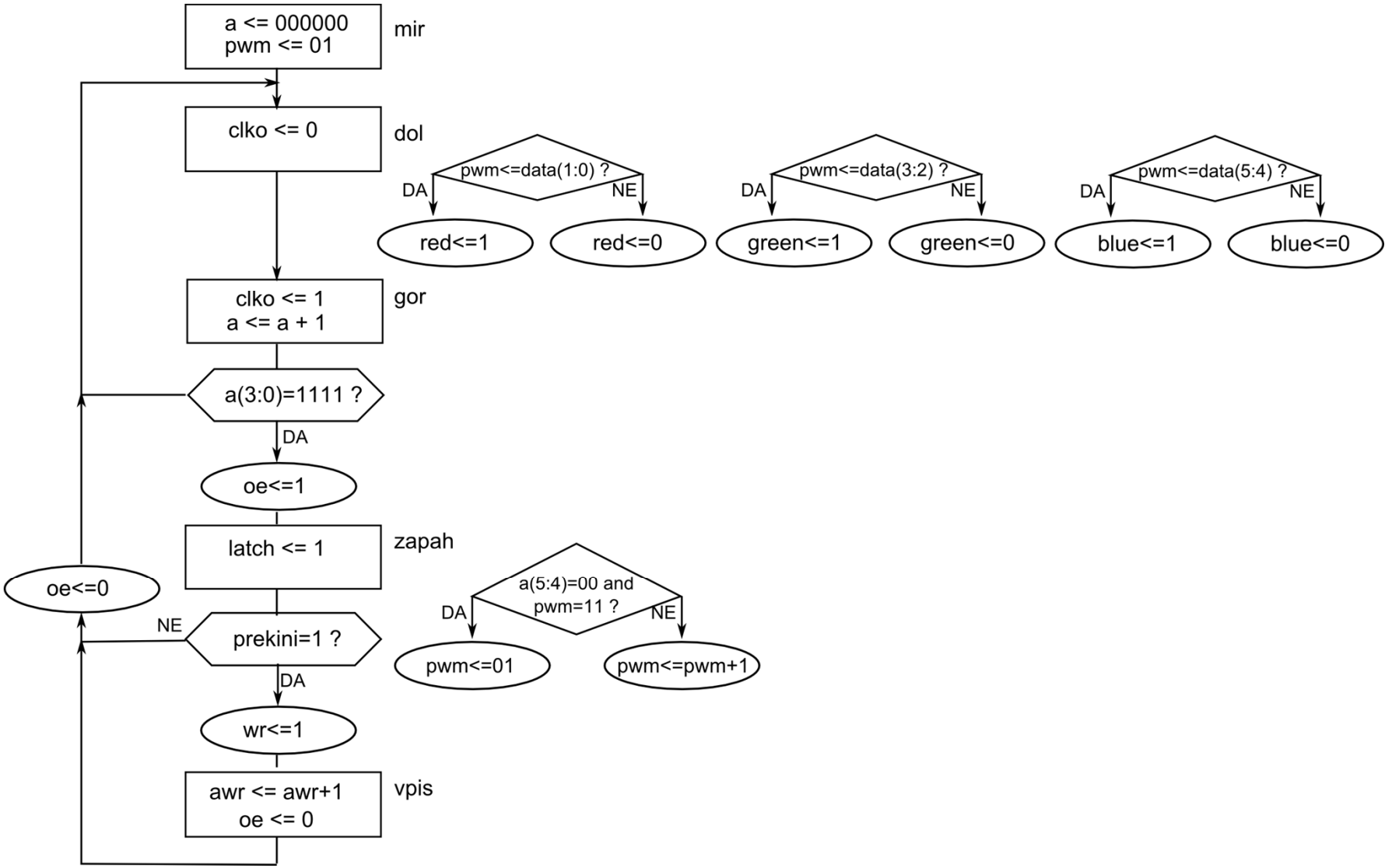
Pred pisanjem kode naredi načrt

- Pred kodiranjem je potrebno poznati strukturo vezja

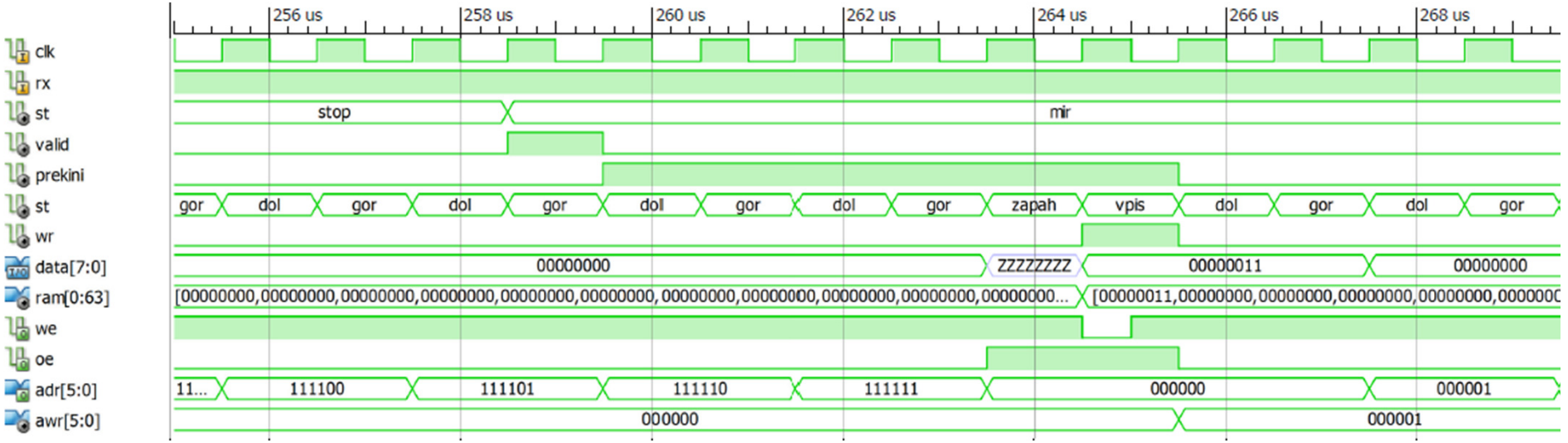


- Nariši diagram podatkovne poti
- Naredi diagram stanj sekvenčnega vezja
  - šele ko so narejeni diagrami začni kodirati

Diagram stanj sekvenčnega vezja: FSM, ASM



Določi časovni potek signalov



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

- ne uporabljaj asinhronih signalov !

```
-- OK for external Reset
process (Clk, Reset)
begin
  if Reset = '1' then
    Q <= '0';
  else
    if rising_edge(Clk) then
      Q <= D;
    end if;
  end if;
end process;
```

```
-- Better
process (Clk)
begin
  if rising_edge(Clk) then
    if Reset = '1' then
      Q <= '0';
    else
      Q <= D;
    end if;
  end if;
end process;
```

2. Pravilno določi seznam signalov v procesu

- v kombinacijskem vezju morajo biti vsi vhodi

```
process (state, long)
begin
  if reset = '1' then
    next_state <= HG;
    start_timer <= '1';
  else
    case state is
      when HG =>
        farm_yellow <= '0';
        if cars = '1' and long = '1' then
          next_state <= HY;
        else
          next_state <= HG;
        end if;
      when HY =>
        farm_yellow <= '0';
        if short = '1' then
          next_state <= FG;
        else
          next_state <= HY;
        end if;
```

```
process (state, reset, cars, short, long)
begin
  if reset = '1' then
    next_state <= HG;
    start_timer <= '1';
  else
    case state is
      when HG =>
        farm_yellow <= '0';
        if cars = '1' and long = '1' then
          next_state  <= HY;
        else
          next_state  <= HG;
        end if;
      when HY =>
        farm_yellow <= '0';
        if short = '1' then
          next_state <= FG;
        else
          next_state <= HY;
        end if;
```

Digital Design with Synthesizable VHDL

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

```
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;
```

```
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;
```

```
dovoljen_prehod <= '1' when luc=zelena else '0';
```

ali:

```
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

```
-- OK

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;
```

```
-- Better

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