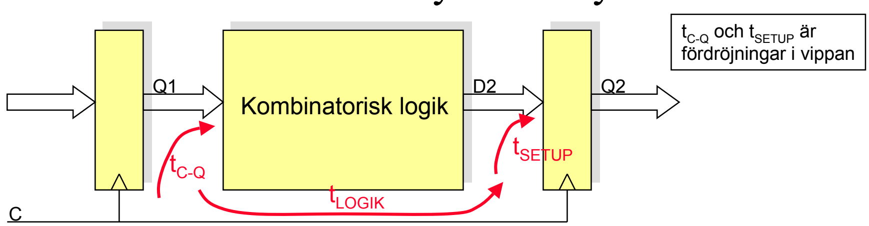
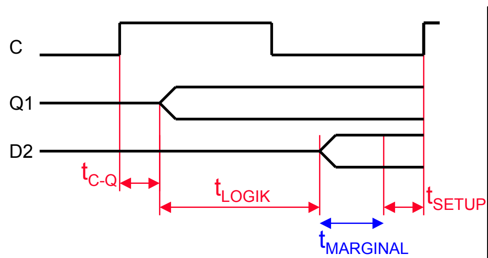
Konstruktion av sekventiell logik

- Innehåll
  - Timing i synkrona system
  - Synkrona processer i VHDL
  - VHDL-kod som introducerar latchar och vippor
  - Initiering av minneselement
  - Mealy- och Moore-maskiner i VHDL
  - Byggblock: räknare, register, RAM/ROM

Timing i synkrona system

- Generell modell för synkront system





En klockcykel är summan av alla fördröjningar:

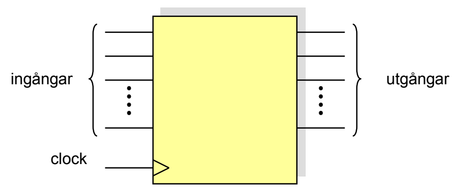
$$T = t_{C-Q} + t_{LOGIK} + t_{SETUP} + t_{MARGINAL}$$

Då vi inte har någon timing-marginal så klockas system på maximal klockfrekvens:

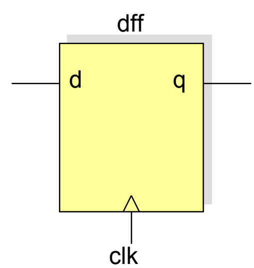
$$f_{\max} = \frac{1}{T_{\min}} = \frac{1}{t_{C-Q} + t_{LOGIK} + t_{SETUP}}$$

Synkrona processer i VHDL

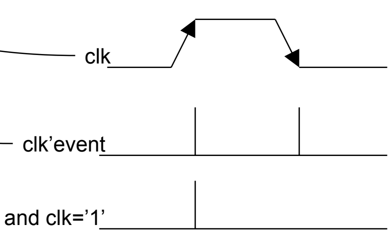
- Synkrona processer
  - Kallas även för klockade processer
  - Alla aktiveras samtidigt, på t.ex positiv klockflank
  - Signal- och variabeltilldelningar i processen resulterar i D-vippor



Exempel: positivt flanktriggad D-vippa i VHDL




```
library ieee;
use ieee.std_logic_1164.ALL

entity dff IS
  port (d, clk : in std_logic;
        q : out std_logic);
end dff;

architecture behavior of dff is
begin
  process(clk)
  begin
    if (clk'event and clk = '1') then
      q <= d;
    else
      q <= q;
    end if;
  end process;
end behavior;
```

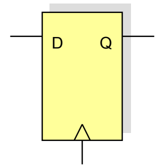


Läser in det nya värdet i vippan

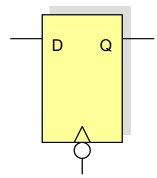
Behåller det gamla värdet

Fler exempel på vippor

```
architecture dataflow of pos_dff is
  q <= d when (clk'event and clk='1') else q;
```

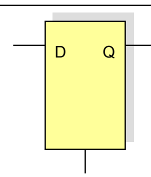



```
architecture dataflow of neg_dff is
  q <= d when (clk'event and clk='0') else q;
```



Exempel på D-latch

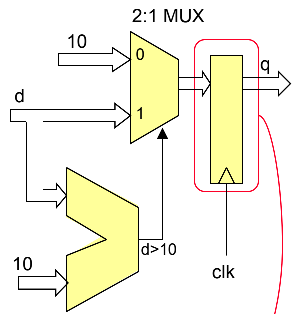
```
architecture dataflow of dlatch is
  q <= d when (clk = '1') else q;
```



Exempel: Kod som inför vippor i konstruktionen

Skriv en funktion som tar in ett 8-bitars tal i ett register om det är större än 10.
Om det är mindre än 10 ska registret innehålla 10.

```
entity load_ge_10 is
  port (
    clk : in     std_logic;
    d   : in     std_logic_vector (7 downto 0);
    q   : out    std_logic_vector (7 downto 0));
end load_ge_10;

architecture rtl of load_ge_10 is
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if d > 10 then
        q <= d;
      else
        q <= conv_std_logic_vector(10,8);
      end if;
    end if;
  end process;
end rtl;
```

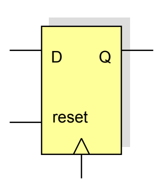


Signal- och variabeltilldelningar leder till införande av register

Initiering av vippor

- Initiering av vippor behövs efter systemstart
- Två typer av initiering
  - Synkron reset/preset
  - Asynkron reset/preset
- Exempel: synkron reset

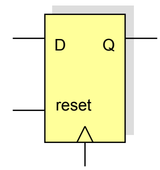
```
if (clk'event and clk = '1') then
  if reset = '1' then q <= '0';
  else q <= d;
  end if;
end if;
```



forts. initiering av vippor

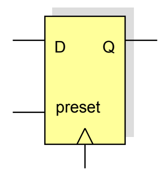
- Exempel: Asynkron reset

```
process (clk, reset)
begin
  if reset = '1' then q <= '0';
  elsif (clk'event and clk = '1') then q <= d;
  end if;
end process;
```



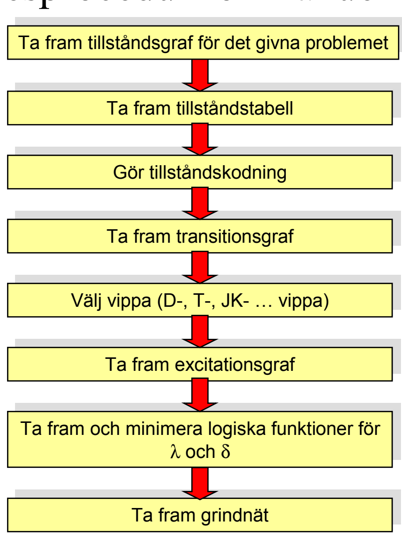
- Exempel: Asynkron preset

```
process (clk, preset)
begin
  if preset = '1' then q <= '1';
  elsif (clk'event and clk = '1') then q <= d;
  end if;
end process;
```



Konstruktion av tillståndsmaskiner

- Syntesprocedur för manuell syntes:

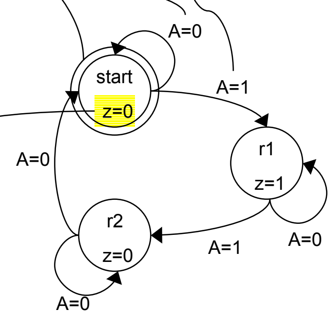


Konstruktion av tillståndsmaskiner

- Syntesprocedur för automatisk syntes

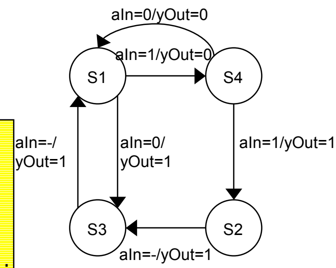
Exempel på tillståndsmaskin i VHDL

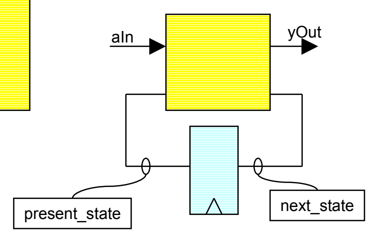
```
architecture rtl of fsm_simple is
  type state_type is (start, r1, r2);
  signal state : state_type;

begin  -- rtl
  update_state : process (clk, reset)
  begin  -- process fsm
    if reset = '0' then
      state <= start;
    elsif clk'event and clk = '1' then
      case state is
        when start  => if A = '0' then state <= start; else state <= r1; end if;
        when r1     => if A = '0' then state <= r1; else state <= r2; end if;
        when r2     => if A = '0' then state <= r2; else state <= start; end if;
      end case;
    end if;
  end process update_state;

  output_logic : process(state)
    begin
      case state is
        when start => z <= '0';
        when r1 => z <= '1';
        when r2 => z <= '0';
      end case;
    end process output_logic;
end rtl;
```



Mealy typ i VHDL

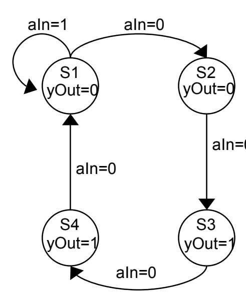
```
architecture mealy of fsm2 is
  type state is (S1, S2, S3, S4);
  signal present_state, next_state: state;
  begin
  process (aIn, present_state) begin
    CASE present_state IS
      when s1 => if (aIn = '1')
        then yOut <= '0'; next_state <= s4;
        else yOut <= '1'; next_state <= s3;
      end if;
      when s2 => yOut <= '1'; next_state <= s3;
      when s3 => yOut <= '1'; next_state <= s1;
      when s4 => if (aIn = '1')
        then yOut <= '1'; next_state <= s2;
        else yOut <= '0'; next_state <= s1;
      end if;
    end case;
  end process;
  process begin
    wait until clk = '1';
    present_state <= next_state;
  end process;
end mealy;
```





Moore typ i VHDL

```
library ieee; use ieee.std_logic_1164.all;
entity fsm1 is
  port (aIn, clk: in std_logic; yOut: out std_logic);
end fsm1;

architecture moore of fsm1 is
  type state is (s1, s2, s3, s4);
  signal present_state, next_state: state;
  begin
    process (aIn, present_state) begin
      case present_state is
        when s1 => yOut <= '0';
          if (aIn = '1') then next_state <= s1
          else next_state <= s2;
        when s2 => yOut <= '0'; next_state <= s3;
        when s3 => yOut <= '1'; next_state <= s4;
        when s4 => yOut <= '1'; next_state <= s1;
      end case;
    end process;
    process begin
      wait until clk = '1';
      present_state <= next_state;
    end process;
end moore;
```



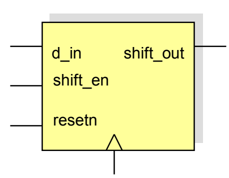
Räknare i VHDL

- Modulo-8 räknare

```
library ieee;
use ieee.std_logic_1164.ALL;
use work.numeric_std.ALL;

entity modulo8 IS
  PORT(clk: in std_logic;
       cnt: buffer unsigned (7 downto 0));
END count8;

architecture rtl of count8 is
begin
  process (clk)
  begin
    if rising_edge(clk) then cnt <= cnt +1;
  end if
end process;
end rtl;
```


Skiftregister i VHDL

```
entity shift_r is
  port (
    clk, resetn, d_in, shift_en : in std_logic;
    shift_out : out std_logic_vector(3 downto 0));
end shift_r;

architecture rtl of shift_r is
  signal shift_reg: std_logic_vector(3 downto 0);
begin
  process (clk, resetn)
  begin
    if resetn = '0' then
      shift_reg <= (others=>'0');
    elsif clk'event and clk = '1' then
      if shift_en='1' then
        shift_reg(3 downto 1) <= shift_reg(2 downto 0);
        -- shift_reg <= shl(shift_reg, "1");
        -- shift_reg <= shift_reg sll 1;
        shift_reg(0) <= d_in;
      end if;
    f;
  end process;

  shift_out <= shift_reg;
end rtl;
```



Alternativa skrivsätt

Minnen i VHDL

- Ett RAM eller ROM kan skapas på två sätt
  - Man använder en array med konstanter
  - Man instansierar en färdig minnesmodul

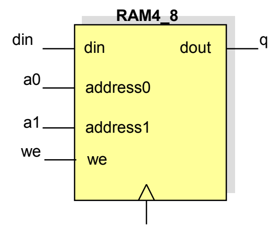
Exempel på 4×8 ROM i VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ROM is

  port (
    address : in  std_logic_vector(1 downto 0);
    dout    : out std_logic_vector(7 downto 0));
end ROM;

architecture rtl of ROM is

  type rom_table is array (0 to 3) of std_logic_vector(7 downto 0);
  constant rom_contents : rom_table := rom_table'("00101111",
                                                  "11010000",
                                                  "01101010",
                                                  "11101101");

begin  -- rtl
  dout <= rom_contents(conv_integer(address));
end rtl;
```


Exempel #1: RAM i VHDL

```
architecture rtl of rmodul is

  component RAM4_8
    port (
      din: in std_logic_vector(7 downto 0),
      address0, address1, we : in  std_logic;
      dout                 : out std_logic_vector(7 downto 0);
      end component;

      begin  -- rtl
        ram1:
          RAM4_8 port map (
            din      => d,
            address0 => a0,
            address1 => a1,
            we       => we,
            dout     => q)
      end rtl;
```

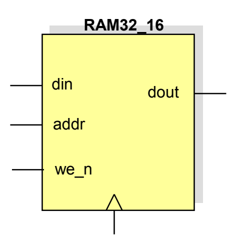
Definiera färdig RAM module från ett bibliotek



Exempel #2: RAM i VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram32_16 is
  port (
    addr      : in  std_logic_vector(4 downto 0);
    clk, we_n : in  std_logic;
    din       : in  std_logic_vector(15 downto 0);
    dout      : out std_logic_vector(15 downto 0));
end ram32_16;

architecture rtl of ram32_16 is

  type ram_type is array (31 downto 0)
       of std_logic_vector(15 downto 0);
  signal ram_array : ram_type;

begin
  process(clk)
  begin
    if clk'event and clk='1' then
      if we_n='0' then
        ram_array(conv_integer(addr)) <= din;
      end if;
    end if;
  end process;

  dout <= ram_array(conv_integer(addr));
end rtl;
```

Definiera en minnesmatris



SLUT på Föreläsning 5

- Innehåll
  - Timing i synkrona system
  - Synkrona processer i VHDL
  - VHDL-kod som introducerar latchar och vippor
  - Initiering av minneselement
  - Mealy- och Moore-maskiner i VHDL
  - Byggblock: räknare, register, RAM/ROM