Høgskulen
på Vestlandet

ELE111 Digitale design

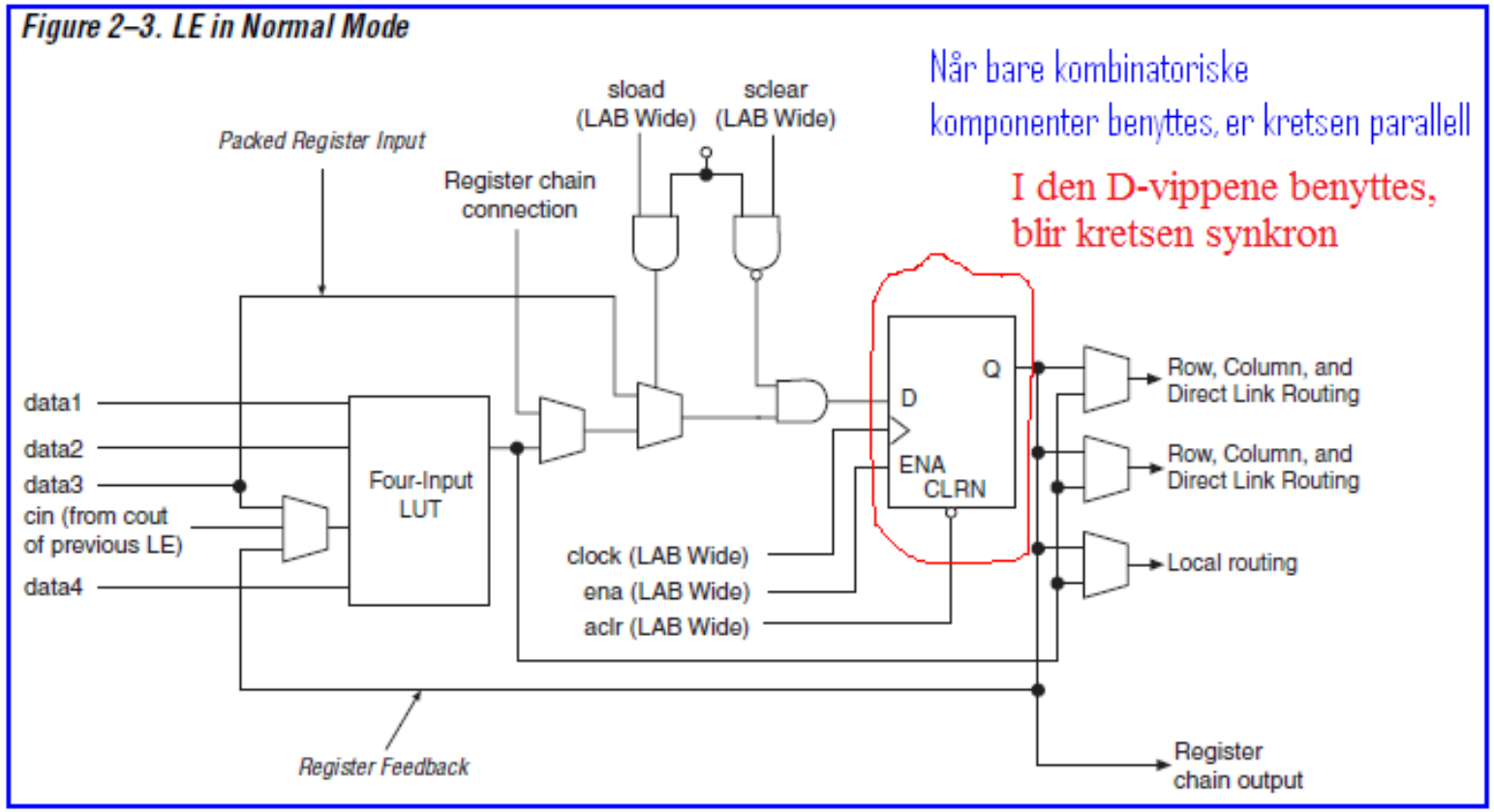
F6_000 Process

```
Latch: process(D,clk)
begin
    if clk = '1' then
        Qa <= D;
    end if;
end process;
```

Eivind Skjæveland
esk@hvl.no

parallell og sekvensiell kode

- I VHDL skiller vi mellom **parallell (Concurrent)** og **sekvensiell** kode
- Sekvensiell kode er inne i ein **process**



Parallell og sekvensiell kode

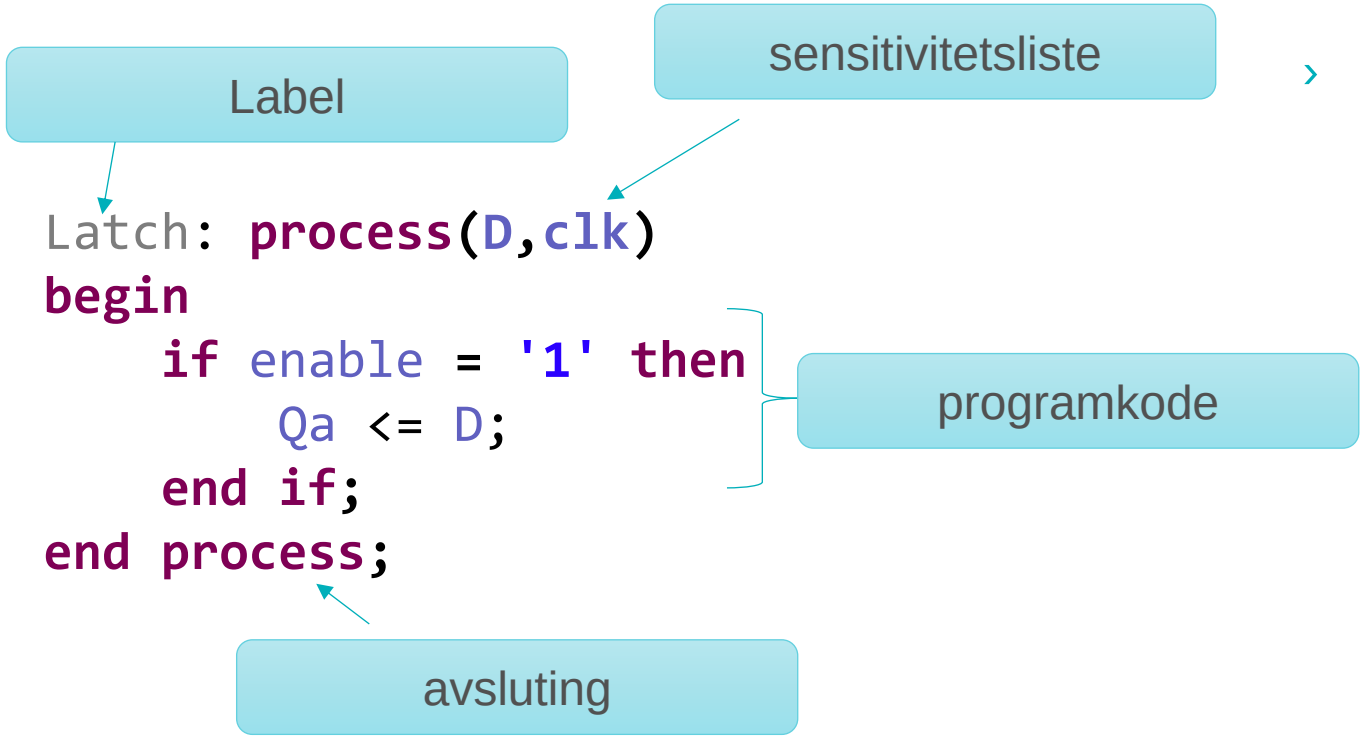
```
architecture RTL of eksempel is
    -- parallell deklarasjonsdel
        -- signal
        -- typer
        -- constant
        -- component
begin
    -- parallell kode,
        --alle kodelinjene blir utført samtidig.
        --rekkefølgen på kodelinjer likegyldig

    process_1 : process (  sensitivitetsliste) is
        -- sekvensiell deklarasjonsdel
            -- variable
            -- constant
            -- typer
    begin
        -- sekvensiell kode
            -- kodelinjene blir utfør ein etter ein.
            -- rekkefølgen på kodelinjer viktig.
    end process process_1;

end architecture RTL;
```

process

- sekvensiell programkode
  - setningane i ein process blir utført i rekkefølge
  - Signal som får nye verdiar i ein process får desse først når processen er ferdig
    - Eller ved WAIT;
  - Alle process-er må ha ein mekanisme for å starta og stoppa processen.
    - WAIT
    - sensitivitetsliste
- process blir plassert i architechture,
  - mellom begin og end

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

entity Lab2_del1 is
    port(
        enable : in std_logic;
        rst : in std_logic;
        D   : in std_logic;

        Qa  : out std_logic;

    );
end entity Lab2_del1;

architecture behavior of Lab2_del1 is

begin
    -- Qa gated D-latch
Latch: process(D, enable)
begin
    if enable = '1' then
        Qa <= D;
    end if;
end process;


end architecture behavior;
```

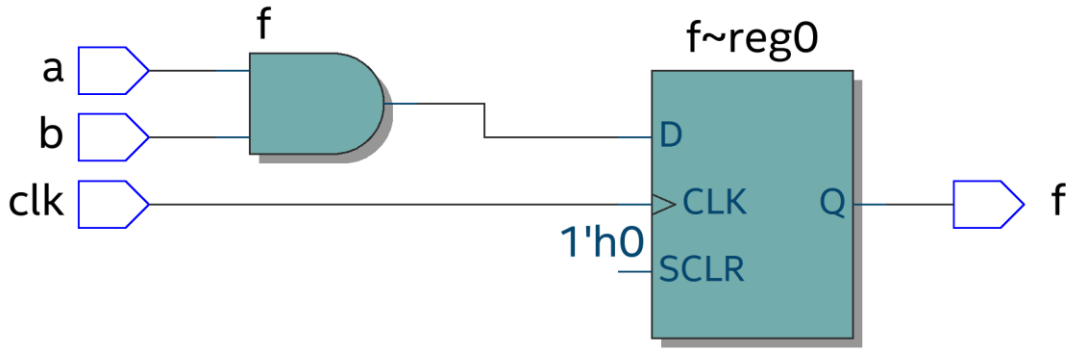
process()



- Oppbygging av process:
  - start med label:,
    - Merkelapp, namn
  - process
  - Sensitivetsliste (i parentes)
    - liste med signal
    - endring i verdi på ei signal i lista får prosessen til å starte
    - kan ha process utan sensitivetsliste
  - Programkode etter begin.
  - avslutt med end process.

synkron process(clk)

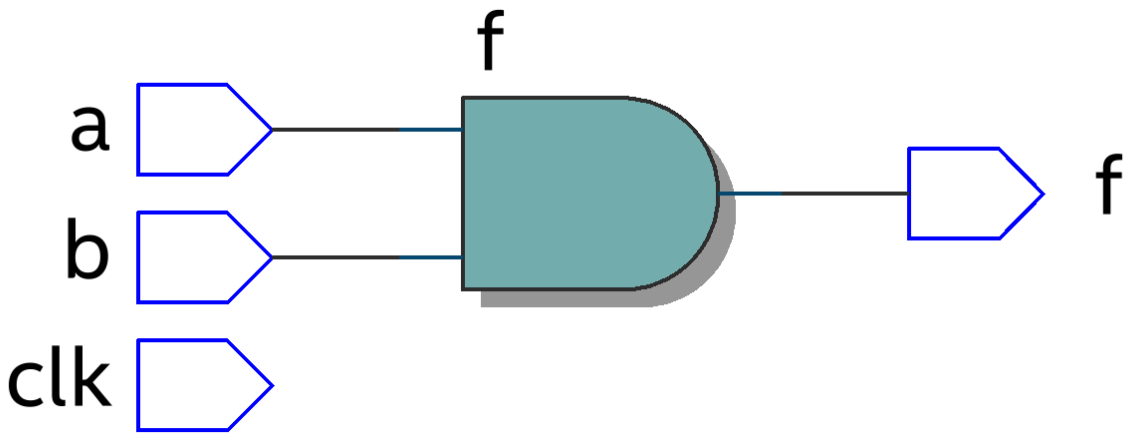
- synkron process
  - `synkron: process(clk)`
  - berre clk på sensitivitetsliste
    - startar på klokkeflanke
    - Lager vipper
      - Alle signal som blir tildelt verdi i ein synkron process blir til ei vippe.




```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity synkron_process is
    port(
        clk : in std_logic;
        a   : in std_logic;
        b   : in std_logic;
        f   : out std_logic
    );
end entity synkron_process;

architecture RTL of synkron_process is

begin
    synkron: process(clk)
    begin
        if rising_edge(clk) then
            f <= a and b;
        end if;
    end process;


end architecture RTL;
```

asynkron process()

- Fleire signal enn klokka med på sensitivitetslista
  - startar når eit av signala endrar verdi
  - Blir vanlegvis til kombinatorisk logikk




```
asynkron : process(a, b)
    begin
        f <= a and b;
    end process;
```

process utan sensitivitetsliste

- Vanlegvis brukt i testbenk
  - Til simulering
  - må ha minst ein WAIT;
  - må spesifisera når processen skal starta og stoppa.
  - WAIT;
    - stoppar processen for godt
  - WAIT for 10 us ;
    - stoppar processen i 10 µs.
    - om vi har WAIT for 10 us på slutten av process
      - Processen vil repetera etter 10 us
  - Signalverdiar blir oppdaterte ved ein wait.

```
p_clk : process is
    begin
        clk <= '0';
        loop
            wait for periode / 2;
            clk <= not clk;
        end loop;
        wait;
    end process p_clk;


    p_reset : process is
    begin
        rst <= '1',
        '0' after 30 ns, '1' after 100 ns,
        '0' after 650 ns, '1' after 700 ns,
        '0' after 950 ns, '1' after 1050 ns,
        '0' after 2500 ns, '1' after 3000 ns;
        wait;
    end process p_reset;



    p_d : process is
        constant vent : time := periode/10;
        constant inputs: std_logic_vector :=
"011101001010010111110000";
    begin
       wait for 125 ns;
       for i in inputs'range loop
           D <= inputs(i);
           wait for vent;
           end loop;
    end process p_d;
```

Lag D-vippe (flop-flop) med process

- Vippe
  - Synkron process
  - inngang : D og clk
  - utgang Q og Q_bar:

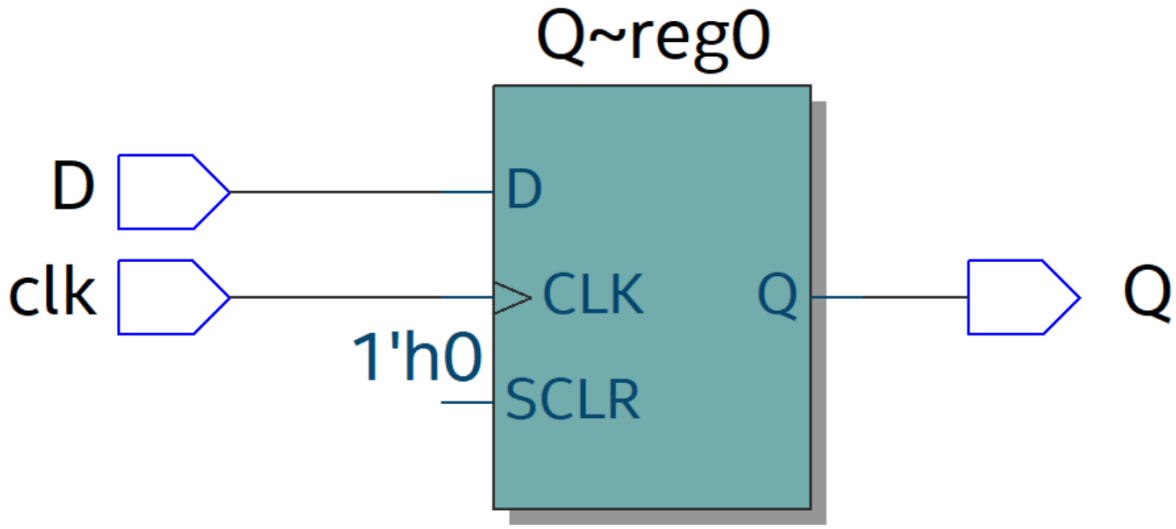
TABLE 7–2

Truth table for a positive edge-triggered D flip-flop.

Inputs		Outputs		
D	CLK	*Q*	$\overline{Q}$	Comments
0	↑	0	1	RESET
1	↑	1	0	SET

↑ = clock transition LOW to HIGH

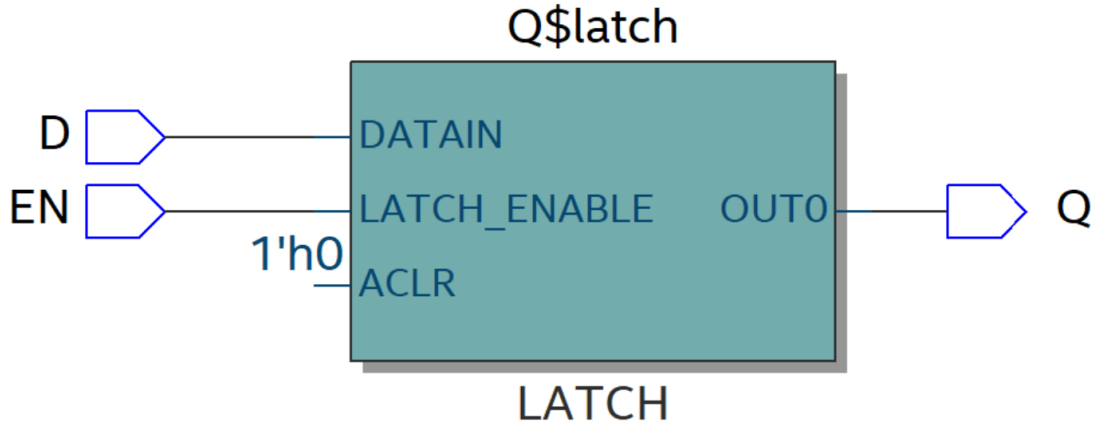
```
D_vippe : process(clk)
    begin
        if rising_edge(clk) then
            Q <= D;
        end if;
    end process;
```



Lås (Latch) med process

```
latch : process(D, EN)
    begin
        if EN = '1' then
            Q <= D;
        end if;
    end process;
```

Inputs		Outputs		
D	*EN*	*Q*	$\overline{Q}$	Comments
0	1	0	1	RESET
1	1	1	0	SET
X	0	Q_0	$\overline{Q}_0$	No change



Process: rekkefølge på statements er viktig.

- Parallell kode:
  - Signalverdiar blir oppdatert umiddelbart
- Process
  - Signal får ny verdi ved `end process`
  - i eksempelet :
    1. framtidig verdi til f blir sett til a;
    2. framtidig verdi til f blir sett til b;
    3. framtidig verdi til f blir sett til c;
    4. Nåverdi til f blir c;
  - linjene `f <= a;` og `f <= b;` får ingen betydning
  -

```
pros_2 : process(clk) is
   begin
       if rising_edge(clk) then
           f <= a;
           f <= b;
           f <= c;
       end if;
   end process pros_2;
```

Sensitivitetsliste – feil og rett bruk i asynkrone design

- bra_process
  - Alle inngangssignal vil starta processen

```
entity asynkron_process is
    port(
        a, b, c_in : in  std_logic;
        c_out, sum  : out std_logic
    );
end entity asynkron_process;
architecture RTL of asynkron_process is

begin
    bra_process : process(a, b, c_in) is
    begin
        c_out <= ((a xor b) and c_in) or (a and b);
        sum   <= (a xor b) xor c_in;
    end process bra_process;

end architecture RTL;
```

- daarlig_process
  - berre a på sensitiviteslista
  - Berre a vil starta processen

```
entity asynkron_process is
    port(
        a, b, c_in : in  std_logic;
        c_out, sum : out std_logic
    );
end entity asynkron_process;
architecture RTL of asynkron_process is

begin
    daarlig_process : process(a) is
    begin
        c_out <= ((a xor b) and c_in) or (a and b);
        sum   <= (a xor b) xor c_in;
    end process daarlig_process;

end architecture RTL;
```

Parallell og sekvensiell kode

Kommandoer som bare kan benyttes den i parallelle delen	Kommandoer som bare kan benyttes i den sekvensiell delen.
• **Process statement** • **When else statement** • **With select statement** • **Signal declaration** • Block statement	• **If- then- else konstruksjoner** • **Case konstruksjoner** • **Variable deklarasjon** • **Variable tildeling** • **Loop statement** • Return statement • Null statement • Wait statement
Tillatt i både parallell og sekvensiell del	
• **Signal tildeling** • **Deklarasjon av typer og konstanter** • **Boolske uttrykk** • Function og procedure calls • Assert statement • After delay • Signal attributes	

Process