LUNDS UNIVERSITET
Lunds Tekniska Högskola

EITF65
Digitalteknik

FÖRELÄSNING 17 – SEKVENSNÄT MED VHDL

Dagens föreläsning

- Beskrivning av tillståndsmaskiner
  - Kombinatorisk del
  - Minnes del
  - Tillståndsuppdatering
  - Nollställning
- Latchar
- Asynkrona signaler

Förra gången

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lejonbur is
        port (
                G1      : in  STD_LOGIC;
                G2      : in  STD_LOGIC;
                FARA    : out STD_LOGIC;
                clock   : in  STD_LOGIC
        );
end lejonbur;

architecture behav of lejonbur is
        constant c1 : STD_LOGIC := '1';
        signal s1 : STD_LOGIC;
begin
        s1 <= G2 and c1;
        FARA <= G1 and s1;
end behav;
```

Objekttyp
STD_LOGIC
STD_LOGIC_VECTOR
INTEGER

Tilldelning av värde för constant

'1' logisk 1:a
1 heltalet 1

Tilldelning av värde för signal

Förra gången

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sig_vector is
   port (
      a : in STD_LOGIC_VECTOR(3 downto 0);
      b : in STD_LOGIC_VECTOR(3 downto 0);
      z : out STD_LOGIC_VECTOR(7 downto 0)
    );
end sig_vector;

architecture behav of lejonbur is
      signal s1, s2 : STD_LOGIC_VECTOR(3 downto 0);
begin
      s1 <= "0010";
      s2(2 downto 0) <= a(1 downto 0) & b(1);
      s2(3) <= '1';

      z <= s2 & s4;
end behav;
```

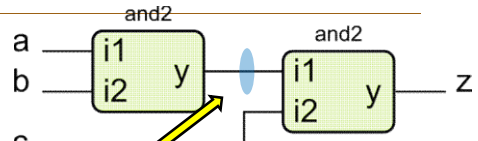
För att sätt alla bitar till ett visst värde
`z <= (others => '0');`

Konkatenera signaler

Observera skillnaden
'1' en bit
"0010" flera bitar

Förra gången

```
entity ex_port_map is
port(a, b, c : in STD_LOGIC;
     z : out STD_LOGIC);
architecture struct of ex_port_map is

  component and2 is
    port(i1, i2 : in STD_LOGIC;
         y : out STD_LOGIC);
  end component;

  signal intern1 : STD_LOGIC;
begin
  comp1:and2 port map(
      i1 => a,
      i2 => b,
      y => intern1);
```

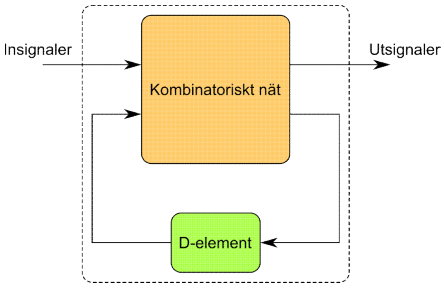


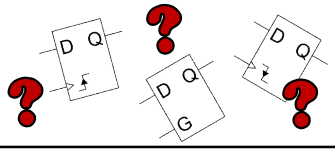

```
  comp1:and2 port map(
        i1 => intern1,
        i2 => c,
        y => z);
end architecture;
```


Sekvensnät - Minneselement

För att konstruera sekvensnät behövs minneselement

- Positivt flanktriggad D-vippa
- Negativt flanktriggad D-vippa
- D-vippa med synkron och asynkron reset
- D-latch (BÖR UNDVIKAS!)





Process - Syntax

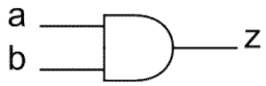
sensitivity list

Etikett för processen

```
proc1: process(..)
begin
  ..
end process;
```

```
and2: process(a, b)
begin
  if a and b then
    z <= '1';
  else
    z <= '0';
  end if;
end process;
```

Signalerna a och b triggar processen



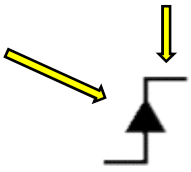
Viktigt att utsignalen alltid får ett värde

Detektera flanker

```
if(clk'event and clk = '1')
```

'1' då det skett en förändring i signalens värde

Signalens värde är '1'



Det går även att använda

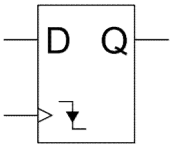
```
rising_edge(clk)
falling_edge(clk)
```

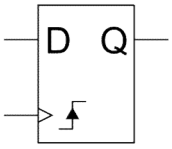
D-vippa

```
architecture arch of FF_neg is
begin
  process(clk)
  begin
    if(clk'event and clk = '0') then
      q <= d;
    end if;
  end process;
end architecture;
```

Sant vid negativ flank

q tilldelas nytt värde






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

Sant vid positiv flank

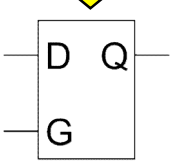
q tilldelas nytt värde

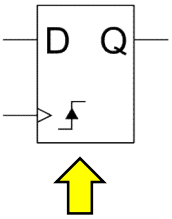
D-latch och D-vippa

```
architecture arch of latch is
begin
  process(g)
  begin
    if(g = '1') then
      q <= d;
    end if;
  end process;
end architecture;
```

Sant vid '1'

q tilldelas nytt värde

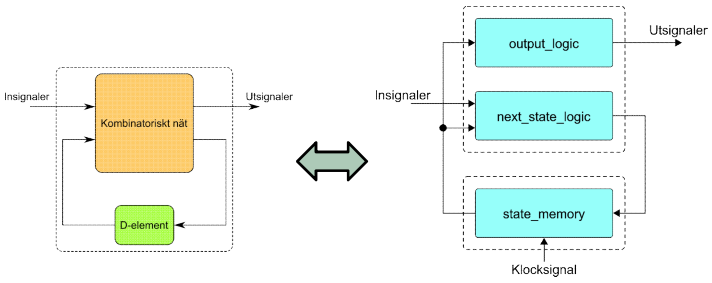





```
architecture arch of FF_pos is
begin
  process(clk)
  begin
    if(clk'event and clk = '1') then
      q <= d;
    end if;
  end process;
end architecture;
```

Sant vid positiv flank

Tillståndsmaskin

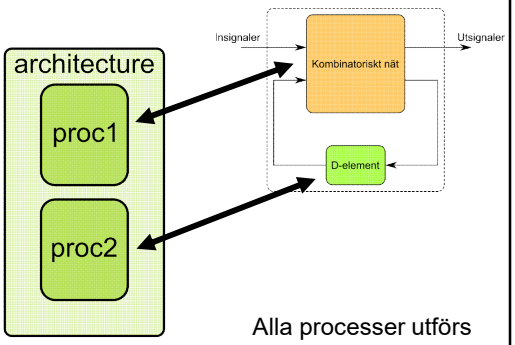


Designen blir enklare att förstå och att felsöka

Process

- Beskriva beteendet av en digital krets på högre abstraktionsnivå
- LIKNAR traditionell sekventiell programmering
- IF-ELSE och CASE satser kan användas

```
..
architecture behav of system1 is
          ..
begin
  proc1: process(..)
  begin
    ..
  end process;

  proc2: process(..)
  begin
    ..
  end process;
end architecture;
```



Alla processer utförs parallellt

Egendefinierade typer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lejonbur is
        port (
                inp     : in  STD_LOGIC;
                o       : out STD_LOGIC);
end lejonbur;

architecture behav of lejonbur is
        type my_type is (on, off);
        signal s1 : my_type;
begin
        if inp = '1'
                s1 <= on;
        else
                s1 <= off;
        end if;
        ...
```

Deklaration av ny signaltyp

Möjliga värden för typen

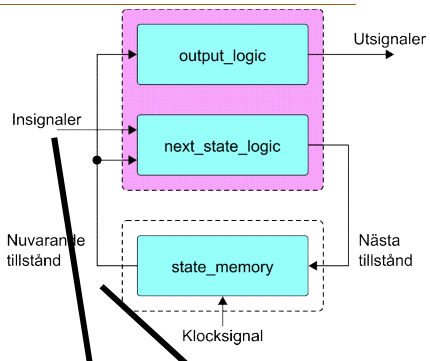
Användbart vid beskrivning av FSM

Kombinatorisk del

Genererar utsignaler och nästa tillstånd

Processen ska triggas av
- Insignaler
- Nuvarande tillstånd

sensitivity list

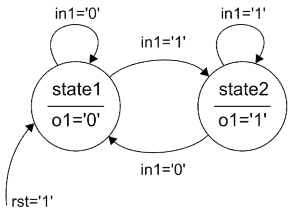



```
komb: process(..)
begin
  ..
end process;
```

```
komb: process(#insignaler#,#nuvarande tillstånd#)
begin
  <nästa tillstånd> <= ..
  <utsignaler> <= ..
end process;
```

Nästa tillstånd

```
komb: process(current_state, in1)
begin
  case current_state is
  when state1 =>
    if(in1 = '1') then
      next_state <= state2;
    else
      next_state <= state1;
    end if;
  when state2 =>
    if(in1 = '1') then
      next_state <= state2;
    else
      next_state <= state1;
    end if;
  end case;
end process;
```

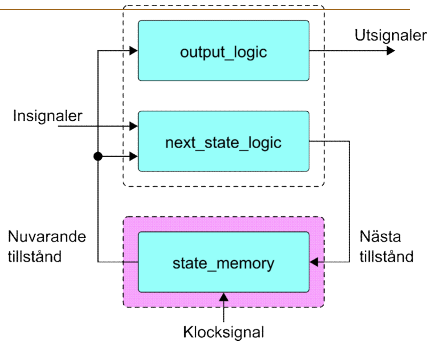


Viktigt att processens utsignaler ALLTID får ett värde!

Minnes del

Generera nuvarande tillstånd

Processen ska triggas av
- Klocksignal
- Resetsignal



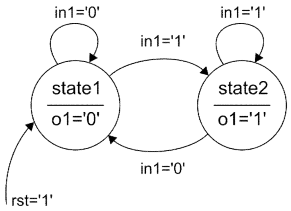

```
mem: process(#Klocksignal#, #reset#)
begin
  if rising_edge(<clock_signal>) then
    <current_state> <= <next_state>;
  end if;
end process;
```

Endast klock- & resetsignal ändrar innehållet i minnet

Tillståndsuppdatering

```
mem: process(clk, rst)
begin
  if rising_edge(clk) then
    if (rst='1') then
      current_state <= state1;
    else
      current_state <= next_state;
    end if;
  end if;
end process;
```

Nollställning av minnet



Om möjligt använd en klocksignal för att uppdatera alla minnen i designen

Om möjligt uppdateras minnen på samma klockflank

Resetsignal

Viktigt att kretsen startar i ett känt tillstånd

Aktiv hög asynkron reset

```
process(rst, clk)
begin
  if(rst = '1') then
    current_state <= res_state;
  elsif rising_edge(clk) then
    current_state <= next_state;
  end if;
end process;
```

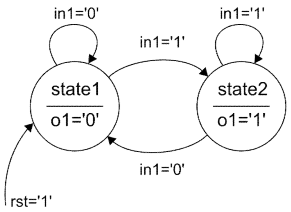
Aktiv hög synkron reset

```
process(clk)
begin
  if rising_edge(clk) then
    if(rst = '1') then
      current_state <= res_state;
    else
      current_state <= next_state;
    end if;
  end if;
end process;
```

Aktiv hög reset då minnen nollställs när `rst='1'`
Aktiv låg reset då minnen nollställs när `rst='0'`

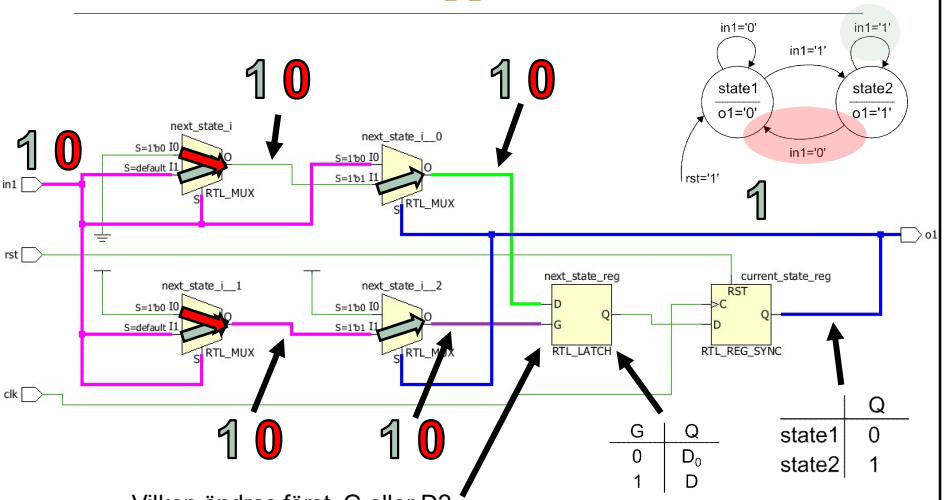
Oavsiktliga latchar

```
komb: process(current_state, in1)
begin
  case current_state is
  when state1 =>
    if(in1 = '1') then
      next_state <= state2;
    else
      next_state <= state1;
    end if;
  when state2 =>
    if(in1 = '1') then
      next_state <= state2;
    end if;
    if(in1 = '0') then
      next_state <= state1;
    end if;
  end case;
end process;
```



Om in1 är av typen STD_LOGIC kan den anta 7 andra värden
⇒ `next_state` får inte alltid ett värde
⇒ Signalen `next_state` blir ett minneselement

Problem som kan uppstå med latchar



G	Q
0	D_0
1	D

	Q
state1	0
state2	1

Vilken ändras först, G eller D?

G ändras först ➔ Q = 1
D ändras först ➔ Q = 0

Unvika latchar

Se till att kombinatoriska signaler alltid får ett värde

Om inget annat anges stanna kvar i tidigare tillstånd

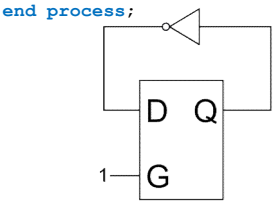
Om inget annat anges

```
komb: process(current_state, in1)
begin

  next_state <= current_state;
  o1 <= '0';
  o2 <= (others => '0');

  case current_state is
  when state1 =>
    if(in1 = '1') then
      next_state <= state2;
  end if;
  when state2 =>
    o1 <= '1';
    o2 <= "100011100";
    if(in1 = '0') then
      next_state <= state1;
    end if;
  end case;
end process;
```

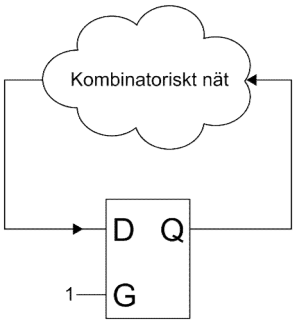

Latch istället för D-vippa

```
mem: process(clk, rst)
begin
  if (clk='1') then
    if (rst='1') then
      current_state <= state1;
    else
      current_state <= next_state;
    end if;
  end if;
end process;
```

clk'event and clk = '1')



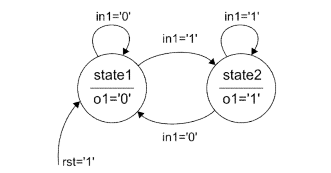
Vilket värde har Q efter att G→0?

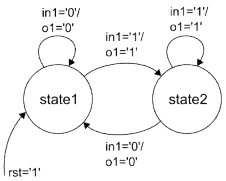


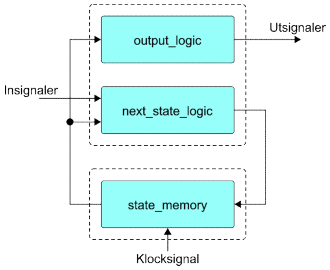
Kombinatoriskt återkopplat nät!

Synthesis (1 warning)
[Synth 8-327] inferring latch for variable 'next_state_reg'

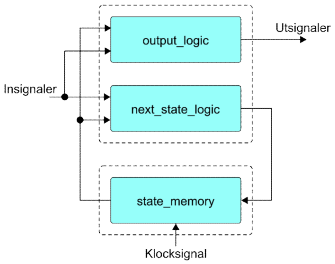
Moore och Mealy







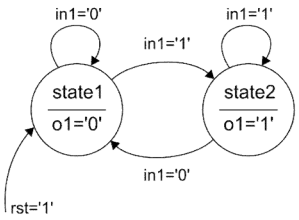
Moore



Mealy

Utsignaler vid Moore

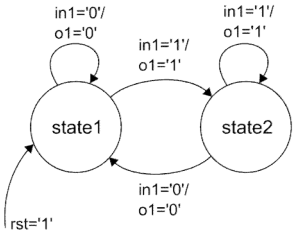
```
output_logic_moore: process(current_state)
begin

  case current_state is
  when state1 =>
    o1 <= '0';
  when state2 =>
    o1 <= '1';
  end case;
end process;
```



Utsignaler beror endast på nuvarande tillstånd

Utsignaler vid Mealy

```
output_logic_mealy: process(current_state, in1)
begin

  case current_state is
  when state1 =>
    o1 <= '0';
    if (in1 = '1') then
      o1 <= '1';
    end if;
  when state2 =>
    o1 <= '1';
    if (in1 = '0') then
      o1 <= '0';
    end if;
  end case;
end process;
```

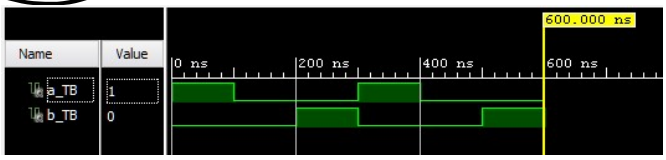


Utsignaler beror på nuvarande tillstånd och insignaler

Generera testsignaler

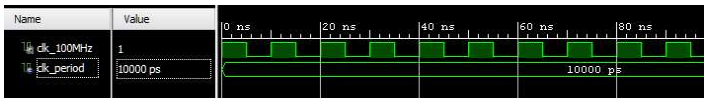
```
library ieee;
use ieee.STD_LOGIC_1164.all;

entity system_TB is
end entity;

architecture arch of system_TB is
    component system is
        port(a, b : in STD_logic;
             f : out STD_LOGIC);
    end component;

    signal a_TB, b_TB, f_TB : STD_LOGIC
begin
    DUT:system port map(
        a => a_TB,
        b = b_TB,
        f => f_TB
    );
```

```
signal_gen:process
begin
    a_TB <= '1';
    b_TB <= '0';
    wait for 100 ns;
    a_TB <= '0';
    wait for 100 ns;
    b_TB <= '1';
    wait for 100 ns;
end process;
end architecture;
```

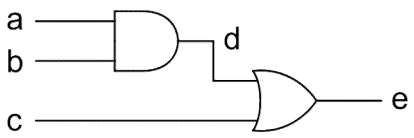


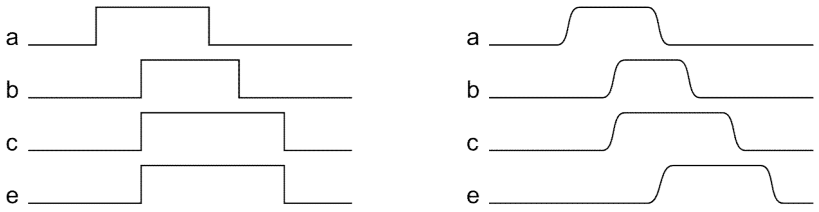
Generera klocksignal

```
library ieee;
use ieee.STD_LOGIC_1164.all;

entity system_TB is
end entity;

architecture arch of system_TB is

    signal clk_100MHz : STD_LOGIC;
    constant clk_period : TIME := 10 ns;
begin
    clk_gen:process
    begin
        clk_100MHz <= '1';
        wait for clk_period/2;
        clk_100MHz <= '0';
        wait for clk_period/2;
    end process;
end architecture;
```

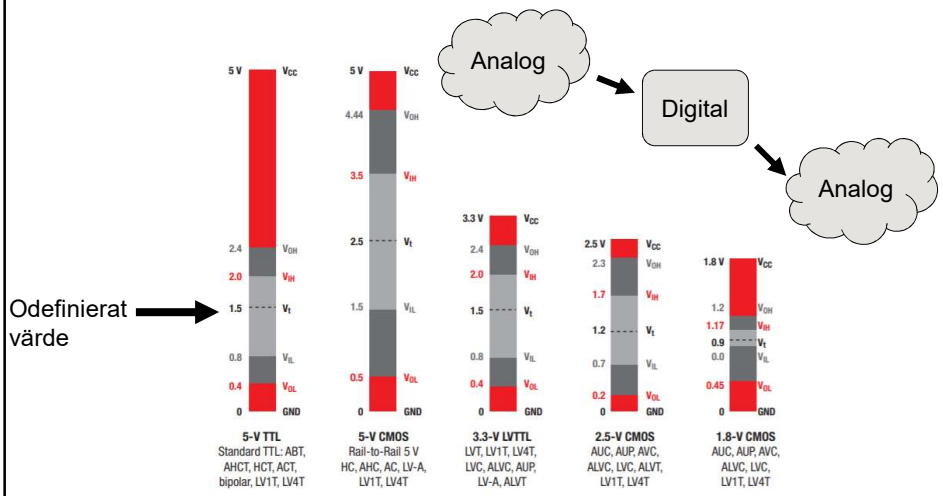


Icke ideala kretsar

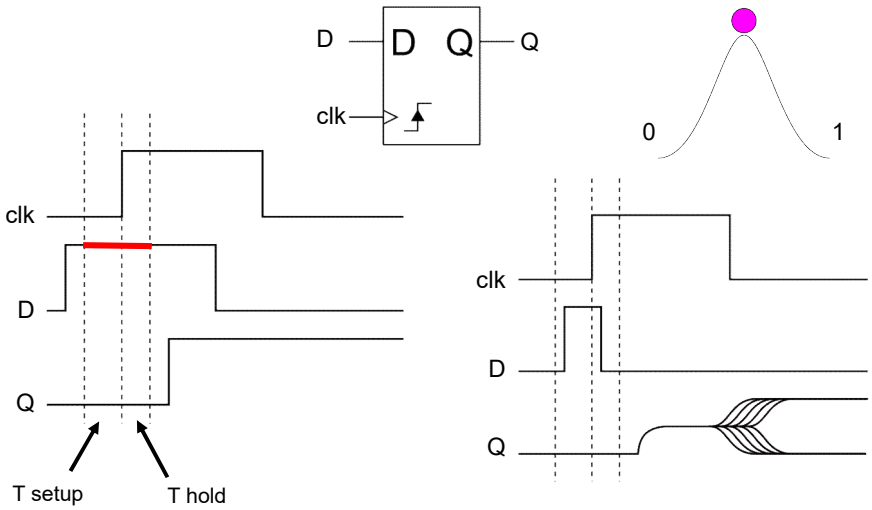




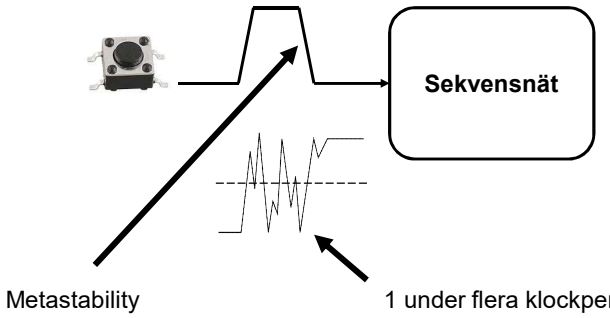
När är insignalen 1



Metastability



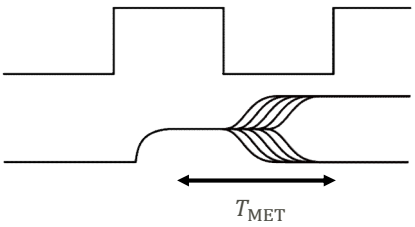
Asynkrona signaler



Metastability

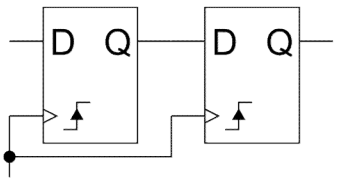
1 under flera klockperoioder

Synkronisering



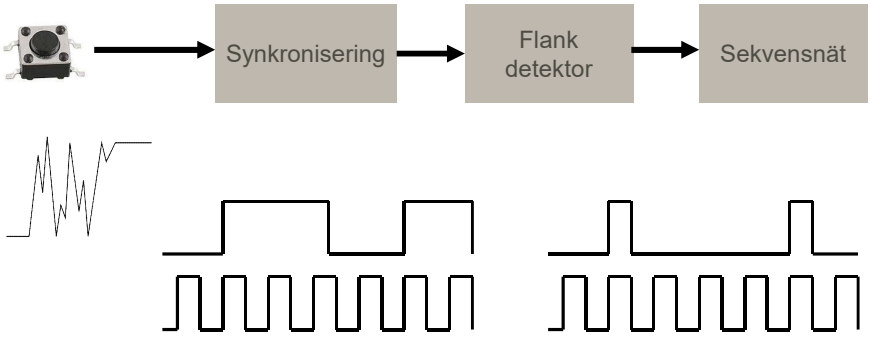
$$MTBF = \frac{e^{T_{\mathrm{MET}} \cdot K_2}}{K_1 \cdot f_{\mathrm{CLK}} \cdot f_{\mathrm{DATA}}}$$

Asynkrona insignaler	MTBF (år) 1 D-vippa	MTBF (år) 2 D-vippor
2	5189	38340
4	2594	19170
8	1297	9585



Fler vippor ökar *MTBF*

Knappavkodning



Gästföreläsning

Den 25 november

8-9 : Fortsättningskurser

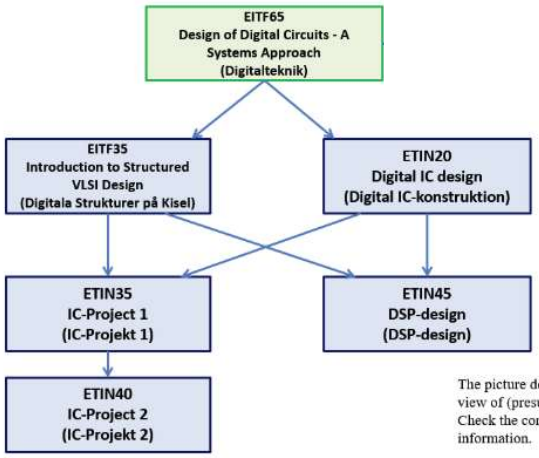
EITF35 - Introduction to Structured VLSI Design

ETIN20 - Digital IC-design

9-10 : Advenica

Cybersecurity solutions

Fortsättningskurser



The picture does not necessarily give a complete view of (presumed) prerequisites for each course. Check the corresponding course plans for such information.

Sammanfattning

- Beskrivning av minneselement
  - Klocksignal
  - Resetsignal
  - Mealy & Moore
  - användardefinierade typer
- Problem med latchar
- Metastability
  - Synkronisering
- Testbench


LUNDS UNIVERSITET
Lunds Tekniska Högskola