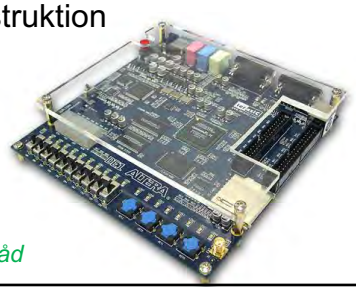
Föreläsning 1
Kursintroduktion

Erik Agrell 2017-08-29

Innehåll:

1. Välkomna till SSY011 Elektriska system
2. Förkunskaper
3. Introduktion till elektronikkonstruktion
4. Test och verifiering



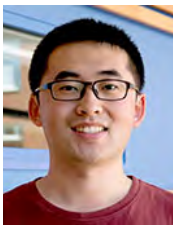
Föreläsningsbilder av Erik Agrell

Tack till kursens tidigare föreläsare
Manne Stenberg och Arne Linde för bilder och råd

1. Välkomna till
SSY011 Elektriska system

Erik Agrell

föreläsningar
kursorganisation
examination

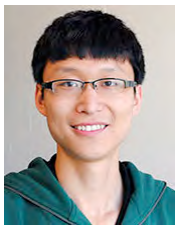


Hao Guo

lab

Cristian Czegledi

lab
inlämningsuppg.



Li Yan

lab
inlämningsuppg.

Syfte

Kursen skall ge erfarenheter av några metoder och verktyg för analys och konstruktion av ***analoga*** och ***digitala*** elektroniksystem.

Lärandemål

- Använda datorbaserade hjälpmedel för konstruktion, analys och simulering av analog och digital elektronik.
- Hantera datorprogram för schemaritning och simulering av analog elektronik.
- Analysera en CMOS-inverterare i Spice.
- Bestämma övre och undre gränsfrekvenser för olika transistorförstärkarsteg och OP-förstärkarkopplingar med hjälp av simulering.
- Dimensionera en ström-spänningsomvandlare.

- Dimensionera signal- och effektförstärkare.
- Dimensionera kaskadkopplade OP-förstärkare för att uppnå önskade frekvensegenskaper.
- Mäta och analysera frekvensegenskaper hos analoga förstärkare och filter.
- Analysera ett återkopplat systems stabilitet och beräkna villkoren för självsvängning.
- Analysera och modifiera befintlig VHDL-kod.

- Hantera datorprogram för simulering och syntes av digital elektronik baserad på VHDL-kod.
- Hantera datorprogram för programmering av FPGA-krets.
- Utifrån en funktionsbeskrivning konstruera en tillståndsgraf för en tillståndsmaskin.
- Konstruera en digital parallell- till serieomvandlare.
- Konstruera en digital serie- till parallellomvandlare.
- Dimensionera och analysera aktiva Butterworthfilter.

Kursmoment

- Laborationer
- Föreläsningar
- Inlämningsuppgifter

Kurshemsida på Pingpong

- Litteratur, föreläsningsbilder
- Meddelanden
- Inlämning av inlämningsuppgifter
- Anmälan till laborationer (gå till *Projektgrupper*)

Har du behörighet till kurshemsidan?

Examination och betygsättning

- Tentamen ger max 50 poäng.
- Laborationen (utförandet, förberedelseuppgifter och rapporten), bedöms med max 4 poäng. Minst 2 poäng krävs för godkänd labkurs.
- Inlämningsuppgifter, som inlämnas i tid, ger max 1 poäng per uppgift
- Bonuspoäng från lab och inlämningsuppgifter får tillgodoräknas vid ordinarie tentamen (men inte senare).
- För godkänd kurs krävs minst 20 poäng på tentamen (inkl ev bonus) och minst 2 poäng på laborationen.
- För betyg 3, 4 och 5 krävs 20, 30 resp. 40 poäng.

Kurslitteratur

- Bengt Molin: Analog elektronik, 2a uppl, Studentlitteratur 2009
- Lars Bengtsson: Elektriska mätsystem och mätmetoder, Studentlitteratur 2012
- Lab-PM, VHDL-kompendium med mera på kursens hemsida

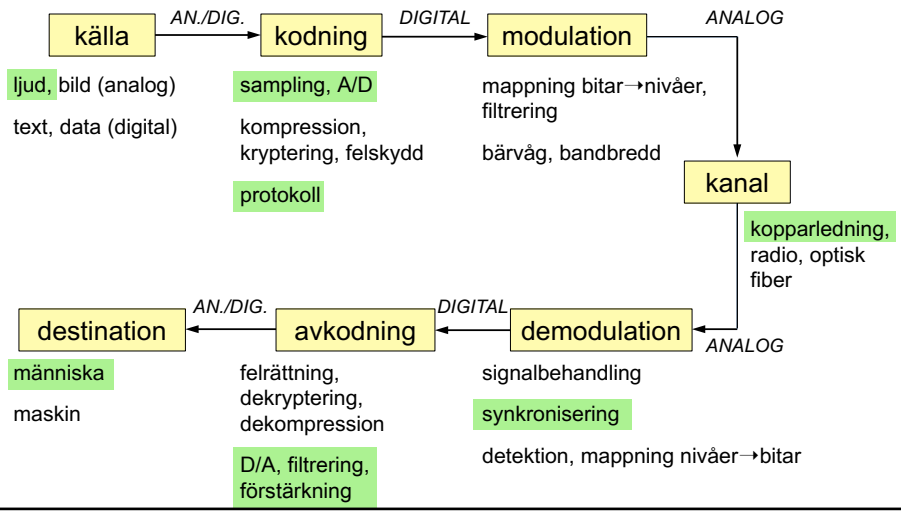
Referenslitteratur

- Johnson, Larsson och Arebrink: Grundläggande digital- och datorteknik, del 1 – Digital teknik, Chalmers
- Hemert: Digitala kretsar, Studentlitteratur
- Sjöholm och Lindh: VHDL, en introduktion, Studentlitteratur

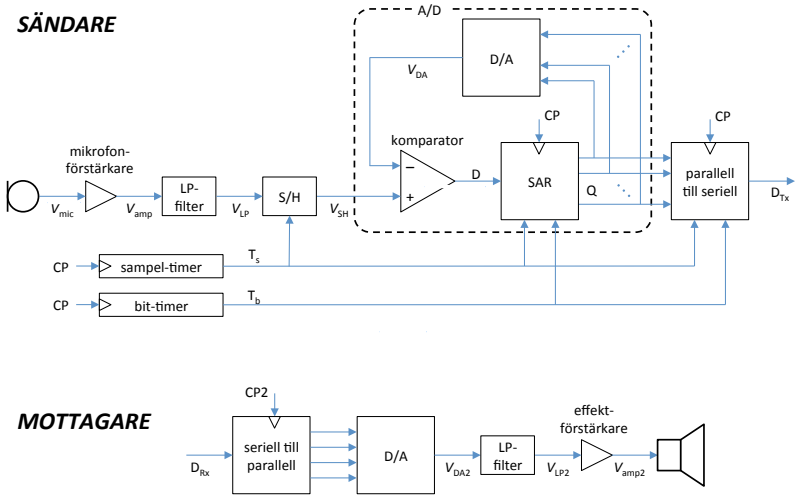
Språk

Svenska	Engelska
Föreläsningar	Laborationer (PM på svenska)
Tentamen	Inlämningsuppgifter

Ett digitalt kommunikationssystem



Laborationer: Konstruktion av ett digitalt ljudöverföringssystem



Upplägg av labkursen

Kontinuerlig inlärning:

- Sex laborationer som bygger på varandra
- Förberedelseuppgifter till varje lab
- Labrapport efter sista labben, en rapport per grupp

Labkursen omfattar formellt 3.5 p ⟷ 95 timmar:

- 24 timmar i labbet
- Resten, 69 timmar, ägnas åt förberedelseuppgifter och rapporten

Rapportering och bonus

- Börja arbeta på rapporten redan från lab 1
- Spara mätresultat, skärmdumpar och foton
- Samarbete uppmuntras, men det är inte tillåtet att kopiera.

Bonuspoäng från labben (max 4 p) får man för
- att lösa problem och kunna förklara förberedelseuppgifterna
- att delta i labarbetet och kunna förklara resultaten
- rapporten

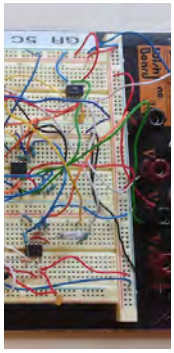
Förberedelseuppgifter

- Labtiden behövs till hårdvaruimplementation
- Beräkningar, konstruktion och simulering behöver vara gjorda före labben
- Fråga gärna om hjälp med förberedelseuppgifterna – före labben

- Om man kommer oförberedd:
  - Små möjligheter att bli klar i tid
  - Mindre bonus
  - I värsta fall underkänt

Labgrupper

- Laborationer görs i par (ej 3)
- Båda skall bidra till gruppens arbete och rapport
- Båda skall kunna svara på förberedelseuppgifter
- Default är att båda får samma poäng, men undantag kan göras
- Labgrupper anmäler sig i Pingpong från onsdag 29 aug kl 13:00
- Kontakta examinator om samarbetet i gruppen inte fungerar



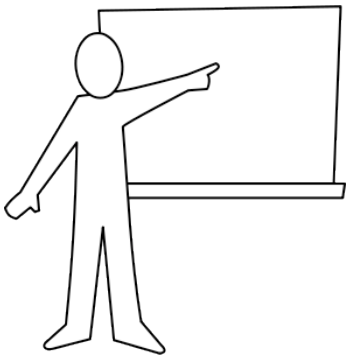
Mjukvara

Programvara	Namn	Ladda ner
VHDL- och FPGA-programmering	Quartus II Web Edition v12.0	altera.com/download
VHDL-simulering	Mentor Graphics ModelSim PE Student Edition (eller QuestaSim 10.4, som är en annan version av samma program)	model.com
Analog krets-simulering	Linear Technology Corporation's LTspice XVII	linear.com/ltspice

Dessa eller liknande verktyg används "överallt" i elektronikindustrin

Prova!
De finns i datasalarna och kan laddas ner till egen dator

Föreläsningar



- 17 föreläsningar av varierande karaktär
- F6 är en övning i LTspice och ModelSim
- F16 är en gästföreläsning från Ericsson
- F17 är repetition

ERICSSON

Inlämningsuppgifter

- 4 inlämningsuppgifter
- Frivilliga
- Förbereder för laboration och tentamen
- Inlämnas individuellt i Pingpong
- Ger bonuspoäng (max 4 p totalt)
- Samarbete uppmuntras, men det är inte tillåtet att kopiera.

Tentamen

- Kursen avslutas med en skriftlig tentamen, som ger max 50 poäng. *Nytt i år: obligatorisk anmälan!*
- Första omtentamen: december 2017. Obligatorisk föranmälan, se tentamensschemat.
- Andra omtentamen: augusti 2018 (skriftlig eller muntlig). Obligatorisk föranmälan till agrell@chalmers.se senast 31 juli. Tentamenstillfället ställs in om ingen har anmält sig 31 juli.
- Hjälpmedel: endast typgodkänd räknare.
- Ett formelblad kommer att bifogas tentamen. Finns på Pingpong.
- Kursinnehållet definieras inte av gamla tentor – viss utveckling görs kontinuerligt.

Deadlines

- Deadlines för inlämningsuppgifter och labrapport anges i kursschemat
- Missad deadline för inlämningsuppgift ⇒ inga poäng
- Missad deadline för labrapport ⇒ max 2p (men fortfarande chans att bli godkänd)
- Inlämning i Pingpong. Kolla att Pingpong verkligen tagit emot inlämningen!
- Examinator kan bevilja förlängd deadline på maximalt 8 timmar, om ansökan kommer in före deadline.
- Förlängning på mer än 8 timmar beviljas inte.

Återkoppling

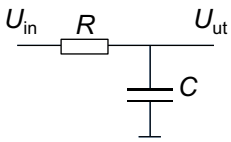
- Konstruktiv återkoppling uppskattas i alla former
- Kursrepresentanter utses nästa vecka

2. Förkunskaper

Exempel på vad ni förväntas kunna från tidigare kurser

- Första ordningens passiva filter:

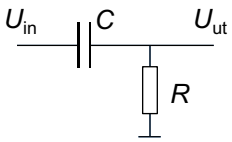
Lågpass (LP)



$$\frac{U_{ut}}{U_{in}} = \frac{1}{1 + j\omega RC}$$

$$f_u = \frac{1}{2\pi RC}$$

Högpass (HP)

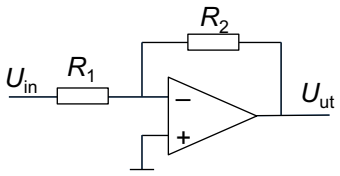


$$\frac{U_{ut}}{U_{in}} = \frac{j\omega RC}{1 + j\omega RC}$$

$$f_{ö} = \frac{1}{2\pi RC}$$

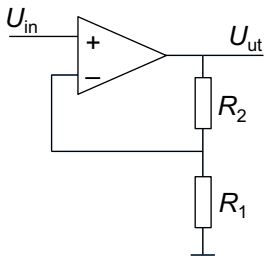
- Enkla operationsförstärkarkopplingar:

Inverterande förstärkare



$$\frac{U_{ut}}{U_{in}} = -\frac{R_2}{R_1}$$

Icke inverterande förstärkare



$$\frac{U_{ut}}{U_{in}} = \frac{R_1 + R_2}{R_1}$$

- Decibel:

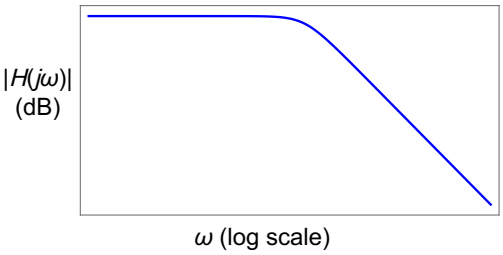
$$\mathrm{dB} = 10 \log \frac{P}{P_0} \quad \text{där } P \text{ och } P_0 \text{ är effekter}$$

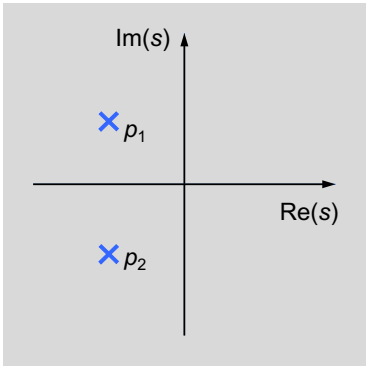
$$\mathrm{dB} = 20 \log \frac{U}{U_0} \quad \text{där } U \text{ och } U_0 \text{ är spänningar}$$

$$\mathrm{dBm} = 10 \log \frac{P}{1\ \mathrm{mW}}$$

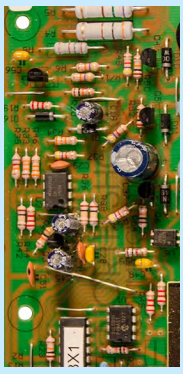
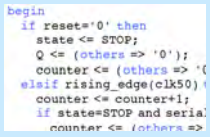
- Överföringsfunktion, Bodediagram, poler och nollställen

$$H(s) = \frac{1}{(s - p_1)(s - p_2)}$$



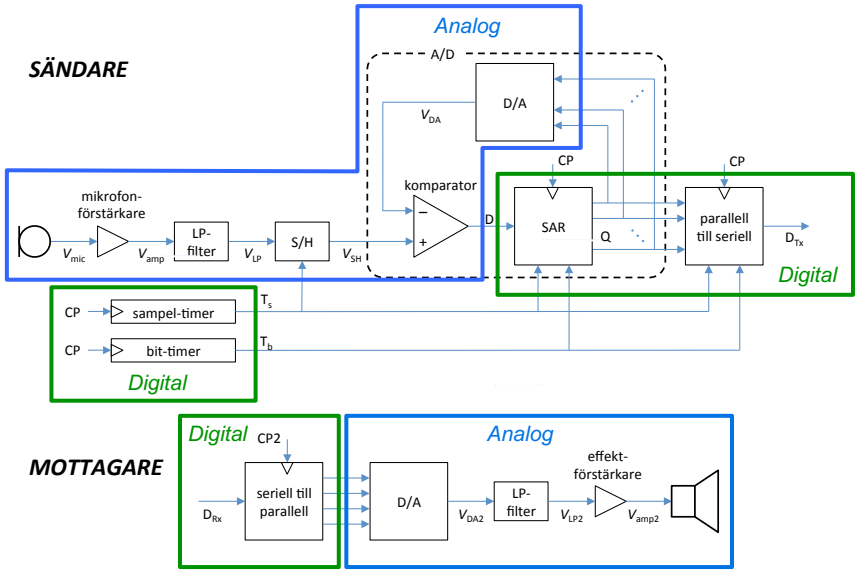


3. Introduktion till elektronikkonstruktion

Analog elektronik	Digital elektronik	Mikrovågs-elektronik	Kraft-elektronik
	 		

Denna kurs

Analog och digital elektronik i labben



FPGA

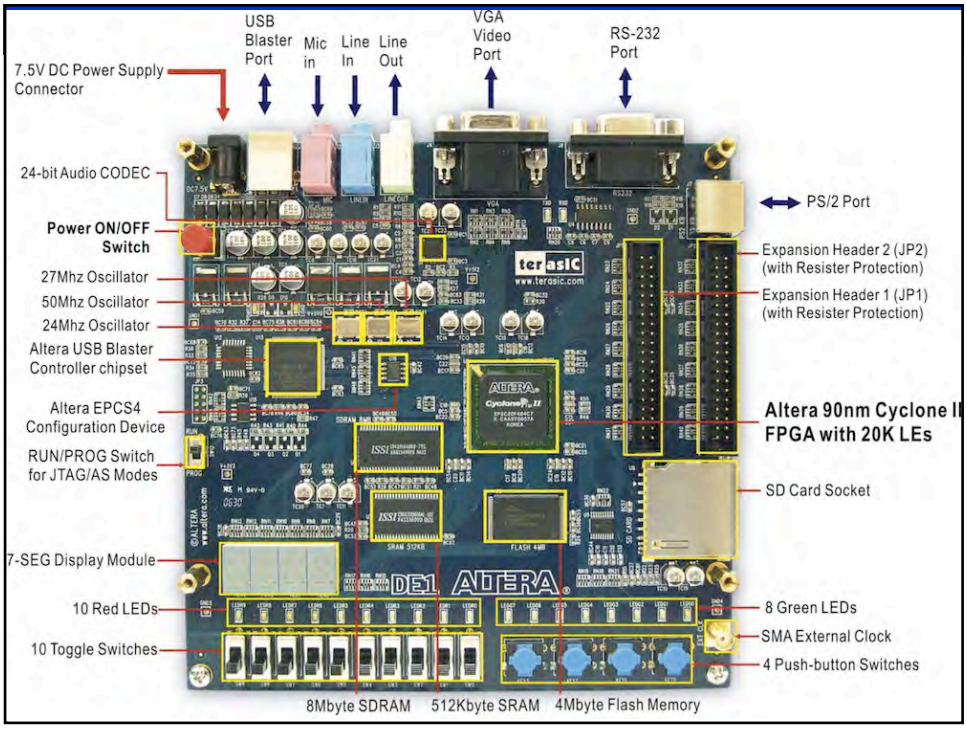
En FPGA (field-programmable gate array) är en flexibel integrerad krets:

- Innehåller en stor mängd generella digitala komponenter: grindar, minne, bussar…
- Förbindelserna mellan dessa enheter kan konfigureras
- Konfigureringen beskrivs i ett HDL-språk (hardware description language)
- Kan konfigureras om hur många gånger som helst

I den här kursen använder vi Alteras FPGA Cyclone II

- Relativt liten (2007)
- Långt kraftfullare än vad vi behöver i den här kursen

- Monterad på ett kort (DE1) med användbar kringutrustning



VHDL

VHDL är ett hårdvarubeskrivande språk som betyder VHSIC Hardware Description Language. Akronymen VHSIC står för Very High Speed Integrated Circuit.

används för

- beskrivning
- simulering
- syntes

Varför VHDL?

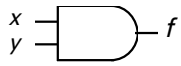
- HDL är metoden för konstruktion av digitala system idag. *(VHDL eller VERILOG)*
- VHDL är ursprungligen framtaget för simulering.
- Simulering och syntes *(kompilering)* till maskinvara är två olika saker!

VHDL-exempel

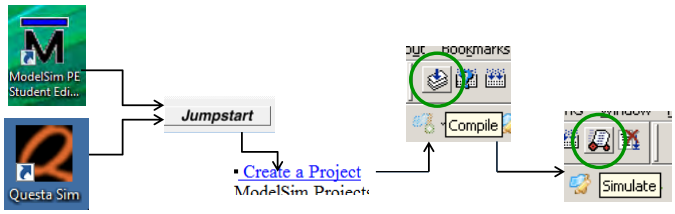
```
library ieee;
use ieee.std_logic_1164.all;

entity and_gate is port (
  x,y: in std_logic;
  f: out std_logic);
end entity;

architecture arch of and_gate is
begin
  f <= (x and y);
end architecture;
```

Vilken komponent beskrivs av denna VHDL-kod?

Svar: och-grind



Demo



4. Test och verifiering

I systemutveckling går vanligen mer resurser till test och verifiering än till själva konstruktionen

"Det är av felsökning man lär sig"

(sagt av Magnus Wisell, elektronikkonstruktör på Ericsson)

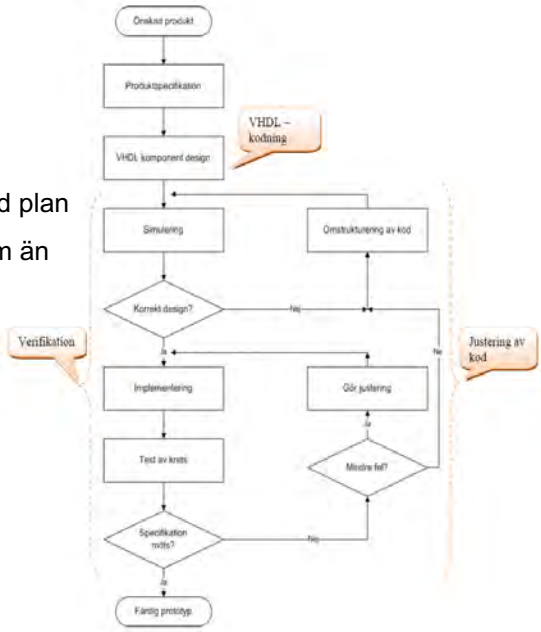
Test och verifiering…

… är en kalkylerad del av utvecklingsprocessen

… görs enligt en fördefinierad plan

… görs ofta av ett annat team än konstruktörerna

… utnyttjar speciella verktyg



När och hur skall man felsöka?

1. Skriv ett långt program, som fungerar direkt

Endast för trollkarlar

2. Skriv ett långt program be någon annan hitta felen

Svårt och ineffektivt

3. Skriv ett långt program som inte fungerar, ta bort små bitar tills det fungerar

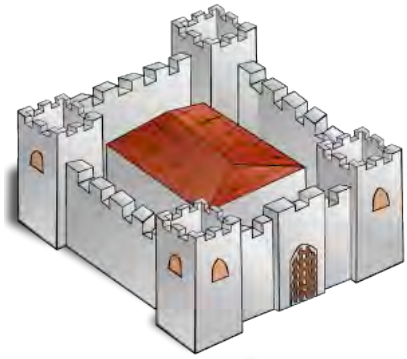
OK

4. Börja med ett program som fungerar, lägg till små bitar och kolla funktionen efter varje tillägg

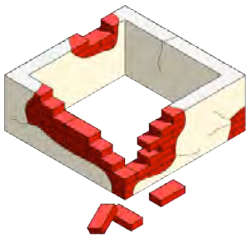
Rekommen-deras!

Felsökning

- Konstruktion och felsökning bör göras parallellt (gäller både hård- och mjukvara)



Bygg ett slott…



…en sten i taget

Att underlätta och förebygga felsökning

- Bygg in testpunkter i konstruktionen
- Bygg modulärt med väldefinierade gränssnitt
- Undvik spagetti (gäller både hård- och mjukvara)
- På kopplingsplattan: använd korta trådar och systematiska färgval

Vilken konstruktion skulle du föredra att testa och felsöka?

Felsökning i labben, allmänt

- Kontrollera att det ni byggde förra gången fungerar innan ni bygger vidare.
- Expandera konstruktionen en liten bit i taget och kontrollera funktionen efter varje tillägg.
- Testa om möjligt nya delar separat innan de inkluderas i hela systemet.
- Koppla tillfälligt bort en bit i taget för att isolera problemet.
- Försök felsöka själv innan du ber om hjälp.

(ur lab-PM)

Felsökning i labben, digital elektronik

- Kolla att VHDL-filens namn är samma som entiteten i VHDL-koden och att den är definierad som Top-Level Entity.
- Minska hastigheten genom att använda långsammare triggsignaler. Lägg in LED så att ni ser vad som händer.
- Kolla speciellt klocka, trigg och tillstånd. Kolla att vektorer har tillräcklig längd för att representera de heltal som behövs.
- För integrerade kretsar, kolla att matning och jord är korrekt anslutna. Kolla kopplingen av jord till DE1-kortet.
- För att testa A/D-omvandling, anslut en DC-nivå till analog input. Variera nivån långsamt och observera digital utsignal (bitar) på LED och/eller oscilloskop.
- Ett syntaxfel kan orsaka många varningar. Bli inte rädd utan försök debugga den första varningen, så kan det hända att flera andra varningar försvinner automatiskt.
- Många kodexempel finns i föreläsningsbilderna, som bör läsas innan ni börjar koda.

(ur lab-PM)

Felsökning i labben, analog elektronik

- Följ signalnivåerna från input till output i respektive modul. Studera signalerna på oscilloskop.
- Mät spänningar i kontrollpunkter och jämför med teoretiskt förväntade värden.
- Om insignalen är OK och utsignalen bottnar (hög eller låg), kontrollera återkopplingsslingan.

(ur lab-PM)

Läs & lös

Läs:

- Molin kap 1 (modeller, dB, begrepp, terminologi)
- Molin kap 4.4–4.5 (kopplingstips, felsökning)

Föreläsning 2
Grunderna i VHDL, ModelSim

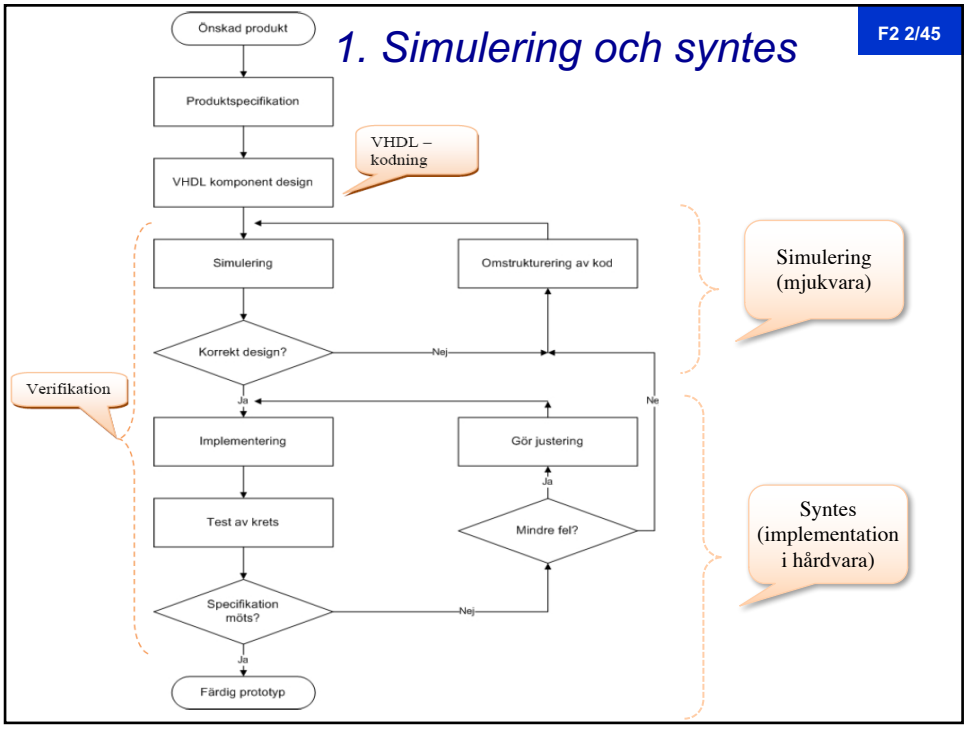
Erik Agrell 2017-08-30

Innehåll:

1. Simulering och syntes
2. VHDLs byggstenar
3. Parallell VHDL
4. Sekventiell VHDL

Bilder av E. Agrell, A. Linde, A. Crayvenn, S. Farinam

1. Simulering och syntes



Arbetsgång i ModelSim

1. Skapa projektmapp
2. Starta ModelSim
3. Create Project, ange namn och mapp
4. New file, ange namn (samma som entiteten)
5. Dubbelklicka på filnamnet
6. Skriv (eller kopiera) VHDL-kod, spara 
7. Kompilera 
8. Felmeddelande? Debugga!
9. Simulera 
10. Markera rätt entitet, under "work"
11. Högerklicka signaler, Add Wave
12. Skriv kommandon: FORCE, RUN

Simuleringskommandon

Sätt x till 1 hela tiden:

```
force x 1
```

Sätt x först till 0, sedan till 1 efter 50 ns, till 0 igen efter 150ns, etc.:

```
force x 0 0ns, 1 50ns, 0 150ns, ...
```

Gör x periodisk med periodtiden 200ns (bra för klocksignaler):

```
force x ... –repeat 200ns
```

Simulera i 500ns:

```
run 500ns
```

(eller bara `run 500` eftersom `ns` är default)

Radera kurvorna och börja om från 0ns vid nästa `run` (som annars fortsätter föregående simulering):

```
restart -force
```

Licensproblem i ModelSim?

```
# vsim -gui
# Start time: 10:05:16 on Sep 02,2014
# ** FATAL ERROR: ModelSim PE Student Edition licensing failure
due to one or more problems with the license key such as:
# - it is not found
# - it has expired
# - it is not for this user
# - it is not for this computer
# - it is not for this version of ModelSim PE Student Edition.
#
# Please go to http://www.model.com and download an updated copy
of the ModelSim PE Student Edition.
# Error loading design

ModelSim>
```

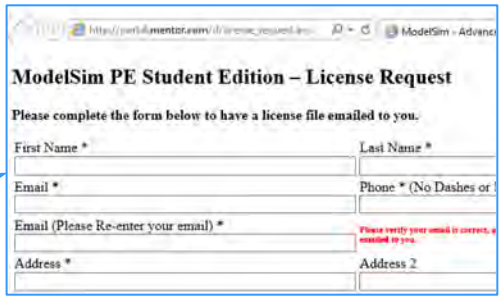

Orsaker och lösningar

```
# Please go to http://www.model.com and download an updated copy
of the ModelSim PE Student Edition.
# Error loading design
```

Orsak 1: Giltig licens saknas

Lösning:

- Beställ en gratislicens i samband med installationen och lägg filen på rätt ställe



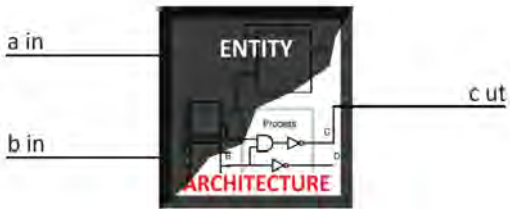
Orsak 2: MentorGraphics tror att du försöker köra två ModelSim på samma licens

Lösningar:

- Avsluta alla andra ModelSim med samma licens
- Om det inte hjälper: Stäng av internet (behövs t.ex. om ModelSim nyligen kraschade)

2. VHDLs byggstenar

- Entiteten beskriver kopplingen till omgivningen
- Arkitekturen beskriver funktionen



- Entiteter och arkitekturer har namn (identifierare)

VHDL-komponent

Vi återvänder till exemplet med och-grinden:

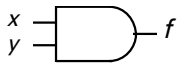
```
-- Simple AND gate
library ieee;
use ieee.std_logic_1164.all;

entity and_gate is port (
  x,y: in std_logic;
  f: out std_logic);
end entity;

architecture arch of and_gate is
begin
  f <= (x and y);
end architecture;
```

kommentar

bibliotek

entitet

arkitektur



Namn i VHDL-kod

- Namn består av bokstäver, siffror och understreck. Måste börja med bokstav.
- Ingen skillnad mellan stora och små bokstäver
- Två bindestreck (--) betyder att resten av raden är kommentar
- Använd inte åäö, inte ens i kommentarer
- Inga mellanslag i filnamn eller sökvägar (path)
- Undvik mycket långa sökvägar

Bibliotek

Ger tillgång till inbyggda definitioner och funktioner

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

Entitet

Definierar gränssnittet: in- och utsignaler

```
entity and_gate is port (
  x,y: in std_logic;
  f: out std_logic);
end entity;
```


Entitetens syntax

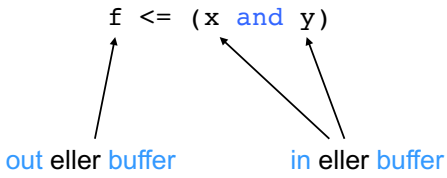
```
entity <komponentnamn> is port (
   <signalnamn>: <riktning> <datatyp>;
   ...
   <signalnamn>: <riktning> <datatyp>);
end entity;
```

Obs parentesens placering!

- Riktning kan vara in, out eller buffer.
- Datatyp kan vara std_logic eller std_logic_vector (Även bit, boolean och integer förekommer.)

in, out och buffer

- in är en ingång. Den kan *inte* tilldelas värden i arkitekturen.
- out är en utgång. Dess värde kan *inte* läsas i arkitekturen.
- buffer är en utgång som är kopplad till ett minneselement. Dess värde *kan* läsas i arkitekturen.



std_logic

Datatypen std_logic kan anta följande värden i simulering:

```
'U' - Uninitialized,        'X' - Forcing Unknown,
'0' - Forcing 0,            '1' - Forcing 1,
'Z' - High Impedance,       'W' - Weak Unknown
'L' - Weak 0                'H' - Weak '1'
'-' - Don't care
```

Bra att känna till att de finns – men vi behöver egentligen bara 0, 1 och U

Arkitektur

Beskriver den interna funktionen

```
architecture arch of and_gate is
begin
  f <= (x and y);
end architecture;
```


Arkitekturens syntax

Arkitekturnamnet är inte viktigt – syns inte utåt

```
architecture <arkitekturnamn> of <komponentnamn> is

   signal <signal_namn>: <datatyp>;
   ...

begin

...

end architecture;
```

Samma som entitetens namn

Alla satser här exekveras samtidigt! Ordningen spelar ingen roll.

Filhuvud

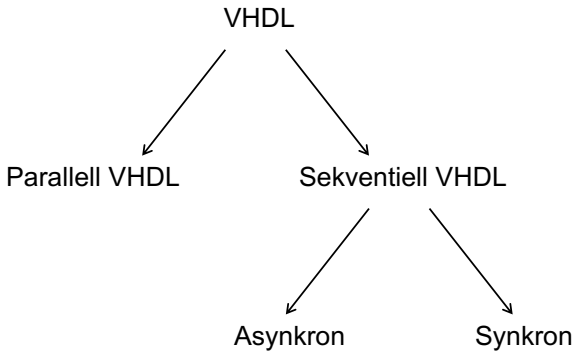
```
----------------------------------------------
-- Company:           Company X
-- Engineer:          Alexander Scott Crayvenn
-- Create Date:       19:38:20 03/30/2009
-- Design Name:       NOR.vhd
-- Module Name:       nor - structure
-- Project Name:      Exempel
-- Target Devices:    Cyclone II

-- Tool versions:     Quartus II, ModelSim PE 6.5
-- Description:       Synthesizable model for
--                    an nor gate
-- Errors:            None
-- Additional Comments:
----------------------------------------------
```


Konventioner

- En filkatalog innehåller ett projekt
- Ett projekt kan innehålla flera VHDL-filer
- En VHDL-fil innehåller en entitet (en komponent)
- En VHDL-fil har samma namn som sin entitet

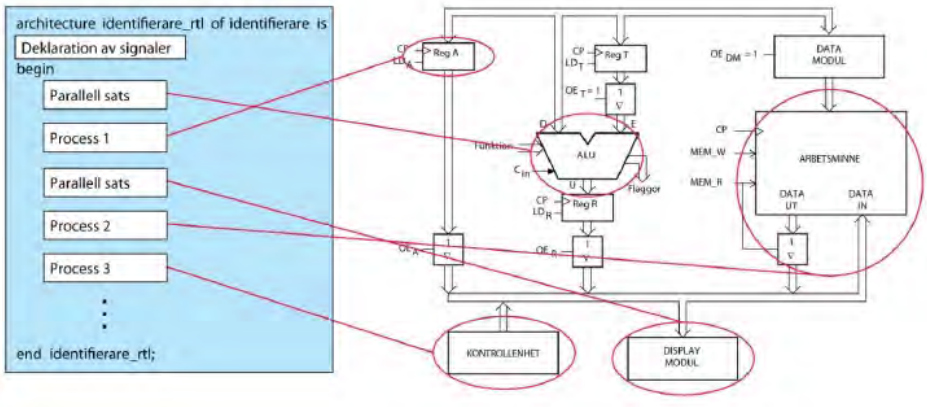
Viktigt!

Olika sorters VHDL-kod



Sorterna kan kombineras i samma komponent

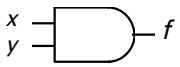
Kombination av parallell och sekventiell VHDL



- Alla satser och processer i architecture exekveras samtidigt (parallellt)

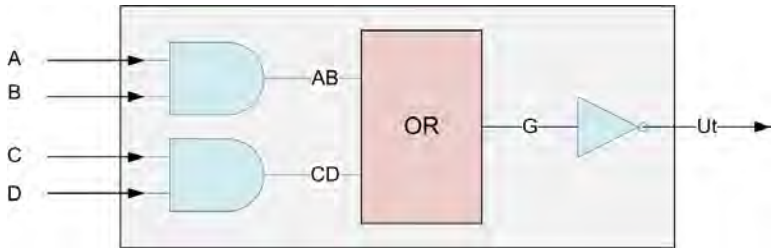
3. Parallell VHDL

Det tidigare AND-exemplet är parallell VHDL (ingen process):

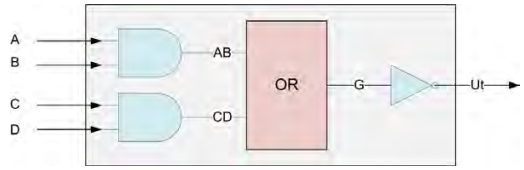



```
library ieee;
use ieee.std_logic_1164.all;

entity and_gate is port (
  x,y: in std_logic;
  f: out std_logic);
end entity;

architecture arch of and_gate is
begin
  f <= (x and y);
end architecture;
```


Exempel med interna signaler



Interna signaler

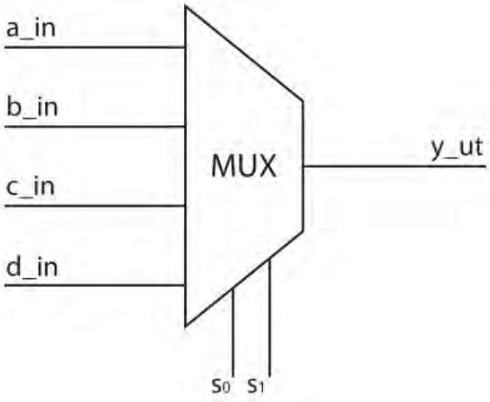



```
library ieee;
use ieee.std_logic_1164.all;

entity AOI is port (
  A,B,C,D: in std_logic;
  Ut: out std_logic);
end entity;

architecture rtl of AOI is
  signal AB,CD,G: std_logic;
begin
  AB <= A and B;
  CD <= C and D;
  G <= AB or CD;
  Ut <= not G;
end architecture;
```

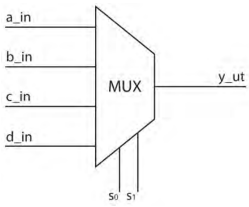
Man kan deklarera flera signaler på samma rad

Finns endast inuti arkitekturen

Två sätt att realisera en multiplexer



Bibliotek och entitet:




```
-- F2: multiplexer
library ieee;
use ieee.std_logic_1164.all;

entity mux is port (
  a_in, b_in, c_in, d_in: in std_logic;
  s0, s1: in std_logic;
  y_ut: out std_logic);
end entity;
```


Metod 1: Grindnivå

Signaltilldelning med tilldelningsoperatorn <= och logiska operatorer:

```
architecture arch of mux is
begin
  y_ut <= (a_in and not s1 and not s0) or
          (b_in and not s1 and s0) or
          (c_in and s1 and not s0) or
          (d_in and s1 and s0);
end architecture;
```

Vektorer

- I stället för en lista av `std_logic`, är det bekvämt att använda vektorer
- Deklareras med `std_logic_vector(... downto 0)` ⇨ MSB först
- Enkla citattecken för bitar ('0', '1') och dubbla för vektorer ("0000").
  OBS: Typografiska citattecken (’0’, “00”) funkar inte. Varning för att kopiera kod från ”smarta” ordbehandlare!
- En tilldelning kan gälla en hel vektor:

```
f <= "11111110"
f <= x"FE"
```

eller delar av den:

```
f(0) <= '1'
f(2 downto 0) <= "010"
```


Metod 2: WITH

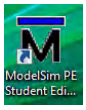
```
architecture arch of mux is
  signal s: std_logic_vector(1 downto 0);
begin
  s <= s1 & s0;
  with s select
    y_ut <= a_in when "00",
            b_in when "01",
            c_in when "10",
            d_in when "11";
end architecture;
```

&: konkatenering

with...select:
s jämförs med
flera värden

Koden ovan innehåller ett vanligt fel. Vilket?

Demo

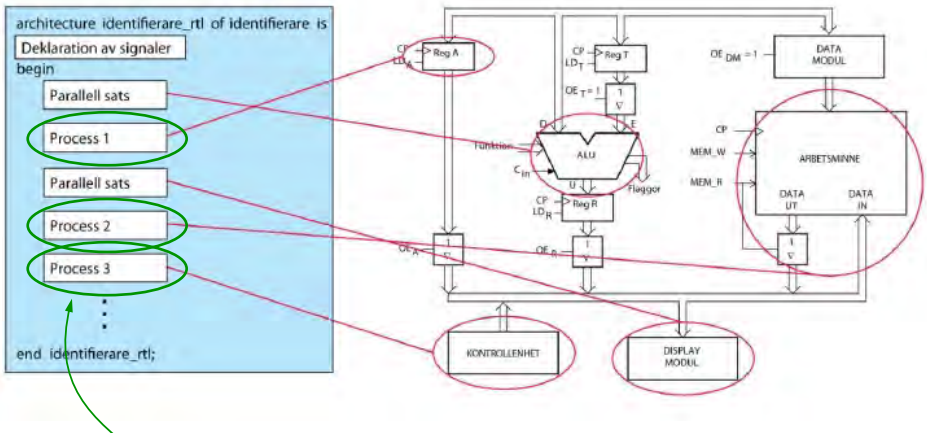


Rättad kod

```
architecture arch of mux is
  signal s: std_logic_vector(1 downto 0);
begin
  s <= s1 & s0;
  with s select
    y_ut <= a_in when "00",
            b_in when "01",
            c_in when "10",
            d_in when others;
end architecture;
```

Vi använder `others` i stället för `"11"` för att täcka alla alternativ

4. Sekventiell VHDL



- Inom varje process exekveras satserna sekventiellt

När exekveras en process?

- I syntes (hårdvara) är komponenterna påslagna hela tiden och anpassar utsignaler efter insignaler
- I simulering är det omöjligt att exekvera hela tiden. Man vill exekvera en process när det behövs och inte annars.

⇒ Användaren behöver tala om för simulatorn när det är dags att exekvera processen

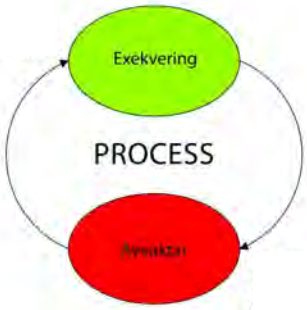
Syntaxen för en process

```
[process_namn:] process (sensitivitetslista)
<variabel namn :data_typ {:= initial_värde;}
begin
<sekventiella satser>
end process [process_namn];
```

- I processens sensitivitetslista ingår de signaler som skall påverka processens igångsättning vid simulering
- Sensitivitetslistan ignoreras vid syntes

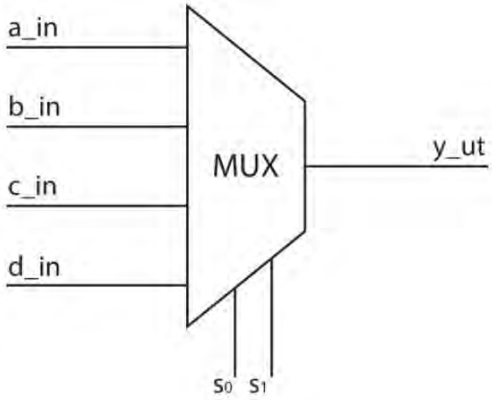
Process och sensitivitetslista

En process simuleras då någon signal i sensitivitetslistan ändras



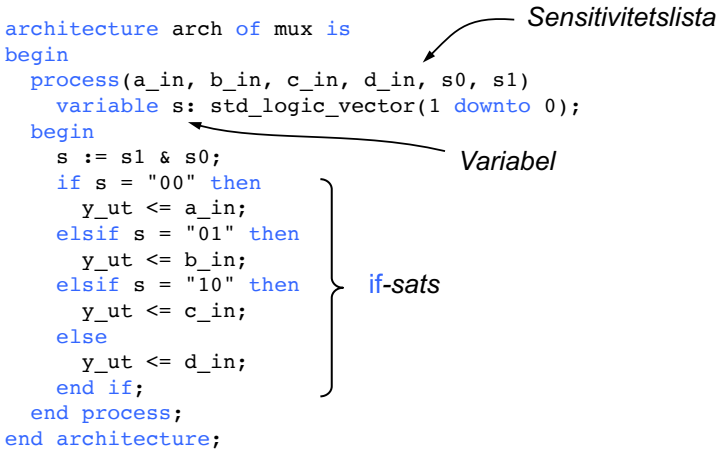
När signalerna i sensitivitetslistan är oförändrade, simuleras inte processen. Utsignalerna ligger kvar oförändrade.

Två sekventiella sätt att realisera MUXen



Metod 3: IF

Vi kan använda samma bibliotek och entitet som förut




```
architecture arch of mux is
begin
  process(a_in, b_in, c_in, d_in, s0, s1)
    variable s: std_logic_vector(1 downto 0);
  begin
    s := s1 & s0;
    if s = "00" then
      y_ut <= a_in;
    elsif s = "01" then
      y_ut <= b_in;
    elsif s = "10" then
      y_ut <= c_in;
    else
      y_ut <= d_in;
    end if;
  end process;
end architecture;
```


Syntaxen för en if-sats

```
if villkor then uttryck;
     else if nytt villkor
          then något annat;
     else nytt uttryck;
end if;
```

- Varje if avslutas med end if
- else if kan sammanskrivas som elsif
- if-satser är sekventiella och får inte placeras i den parallella delen av VHDL-koden

Signaler och variabler

Signaler

- deklareras i den parallella delen
- används både i parallella och sekventiella delen
- tilldelas med <=
- *i parallell kod:* uppdateras hela tiden
- *i sekventiell kod:* uppdateras först då processen avslutas
- I hårdvara realiseras signaler som ledningar

Variabler

- deklareras i den sekventiella delen (process)
- används i den sekventiella delen
- tilldelas med :=
- uppdateras när den tilldelas värde
- existerar inte utanför processen
- I hårdvara realiseras variabler som register

Metod 4: CASE

```
architecture arch of mux is
begin
  process(a_in, b_in, c_in, d_in, s0, s1)
    variable s: std_logic_vector(1 downto 0);
  begin
    s := s1 & s0;
    case s is
      when "00" =>
        y_ut <= a_in;
      when "01" =>
        y_ut <= b_in;
      when "10" =>
        y_ut <= c_in;
      when others =>
        y_ut <= d_in;
    end case;
  end process;
end architecture;
```

case-*sats*

Syntaxen för en case-sats

case är den sekventiella motsvarigheten till with

```
CASE selectorSignal IS
   WHEN value1 =>
      sequential code;
   WHEN value1 =>
      sequential code;
  [WHEN value2 =>
      sequential code;]
  [WHEN others =>
      sequential code;]
END CASE [Case label];
```

Om when*-satserna inte täcker alla möjliga värden för* `selectorSignal` *använder vi* others

Alla when-satser i samma case har samma prioritet

Exekvering i en process

- Inom varje process exekveras koden sekventiellt
- Endast den sista tilldelningen av varje signal i en process räknas
- Använd variabler i processer och tilldela motsvarande signal ett värde på slutet (”skuggvariabler”)

Exempel: NAND

```
architecture arch of nand_gate is
begin
  process(x,y)
  begin
    f <= x and y;
    f <= not f;
  end process;
end architecture;
```

Vi vill invertera resultatet. Varför fungerar inte detta?

Svar: `f` *har här samma värde som när processen startade – oavsett vad som hände tidigare i processen!*

Rättad kod

```
architecture arch of nand_gate is
begin
  process(x,y)
  variable z: std_logic;
  begin
    z := x and y;
    f <= not z;
  end process;
end architecture;
```

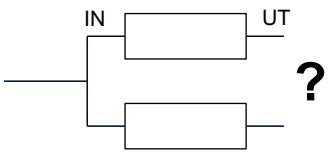
`z` *är en variabel, ingen signal*

Signalen `f` *tilldelas endast en gång. OK!*

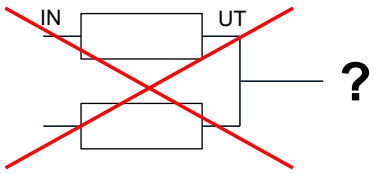
Fast det finns minst två enklare sätt att realisera en NAND-grind

Koppla ihop utgångar?

Exempel: Om man bygger elektronik i hårdvara, är det OK att kombinera kretsar så här?



OK



Fel

- Man kopplar inte ihop utgångarna från två kretsar!
- Det gäller i VHDL också: två satser/processer får inte ha samma utsignaler.

Sammanfattning

Parallell VHDL:

- I architecture, utanför processer
- All kod exekveras samtidigt, ordningen är oväsentlig
- Viktigast är att kunna tilldelningar (<=)

Sekventiell VHDL:

- Sker i processer
- Allt exekveras sekventiellt inom processer – om man använder variabler
- Alla processer exeveras samtidigt

Läs & lös

Läs:

- Lab-PM: inledning och kapitel 1
- Kretskonstruktion med VHDL: avsnitt 1–3.4 och 10.1–10.3

Lös:

- Simulera en XOR-grind i ModelSim genom att följa arbetsgången på bild 3 och modifiera koden på bild 20.
- Inlämningsuppgift 1 (deadline 8 sept)
- Lab-förberedelseuppgift 1.1

Anmälan till labgrupper öppnar idag kl 13:00 i Pingpong

Föreläsning 3
VHDL och FPGA-programmering

Erik Agrell 2017-09-01

Innehåll:

1. FPGA-teknik
2. Synkron VHDL
3. Labkortet DE1
4. Mjukvaran Quartus

Bilder av E. Agrell, A. Linde, A. Crayvenn, S. Farinam, C. Fougstedt

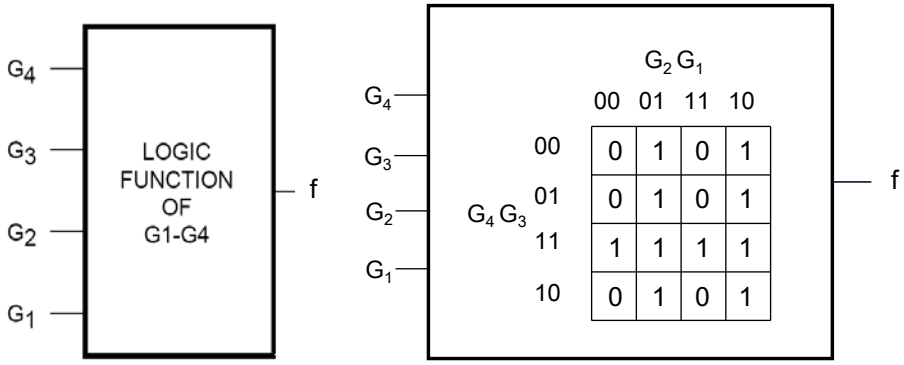
1. FPGA-teknik

En field-programmable gate array (FPGA) består av…

- massor av logikgrindar
- massor av minnesceller (RAM)
- nät av ledningar (interconnects)
- massor av spänningsstyrda kopplingar mellan ledningar
- samt i de mer avancerade modellerna:
  - multiplikatorer
  - processorer (Power PC)
  - AD/DA-omvandlare
  - faslåsta slingor (phase-locked loop, PLL)
  - m.m.

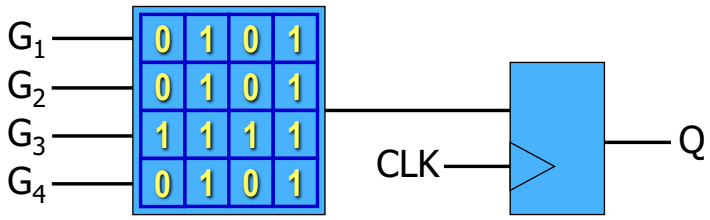
Grindarna i en FPGA

- Grindarna realiseras som en liten tabell (look-up table, LUT)
- Kan ses som ett Karnaugh-diagram
- Kan realisera alla tänkbara grindar



Logikelement

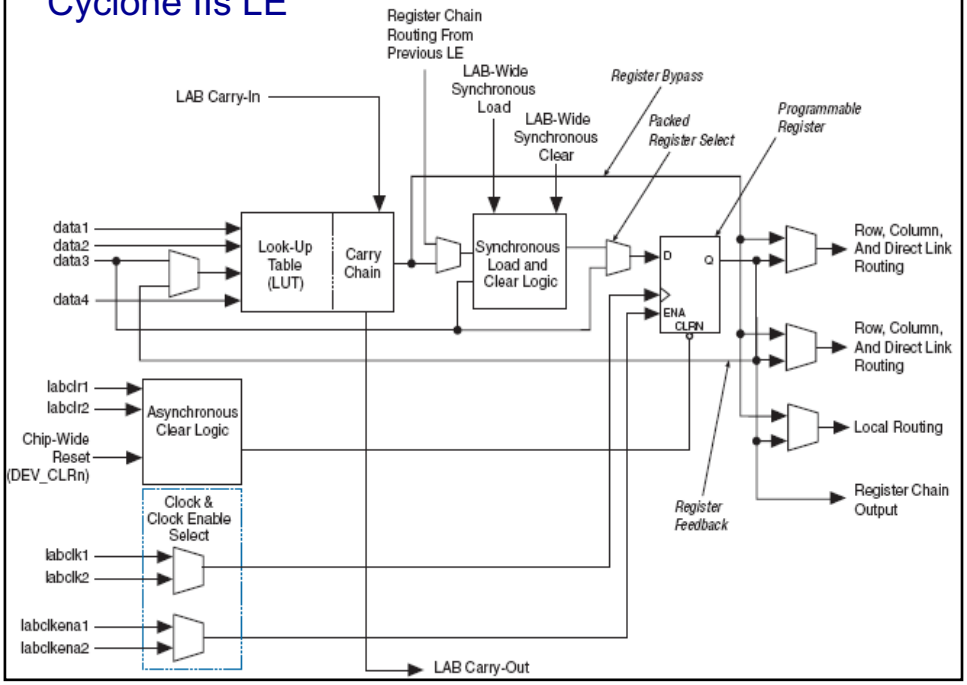
- Kombinationen av grind (LUT) och vippa kallas logikelement (LE)



(förenklad bild – ett LE innehåller normalt även andra komponenter)

- Storleken hos FPGAer mäts i antal LE

Cyclone IIs LE



Labbets FPGA

Cyclone II 2C35 FPGA

- 18,752 LEs
- 52 M4K RAM blocks
- 240K total RAM bits
- 26 embedded multipliers
- 4 PLLs
- 315 user I/O pins
- FineLine BGA 484-pin package

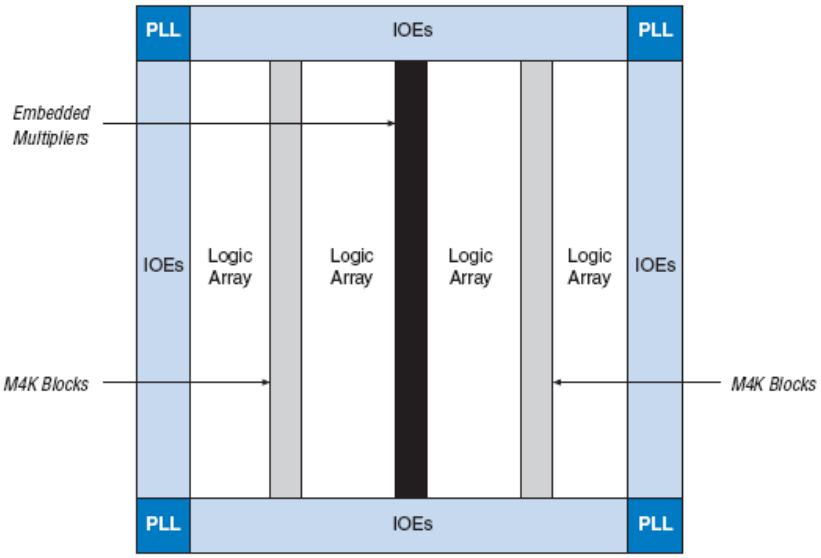
Cyclone är lågprismodellen

Finns nu i version V

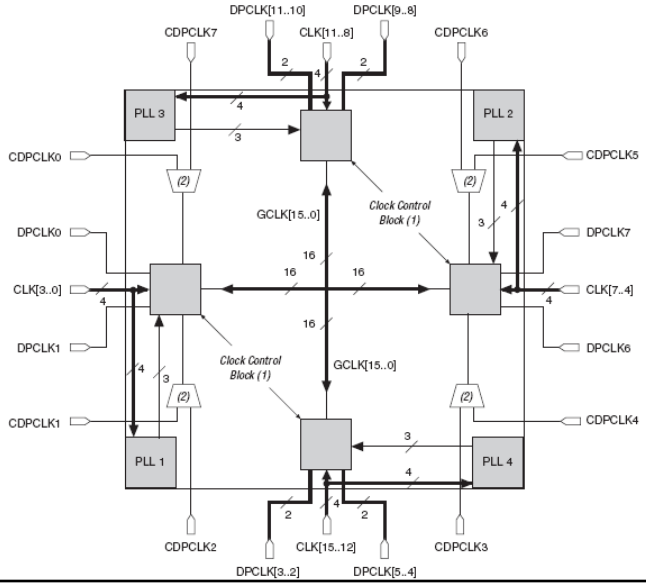
Från ca 300 kr/styck

Cyclone II

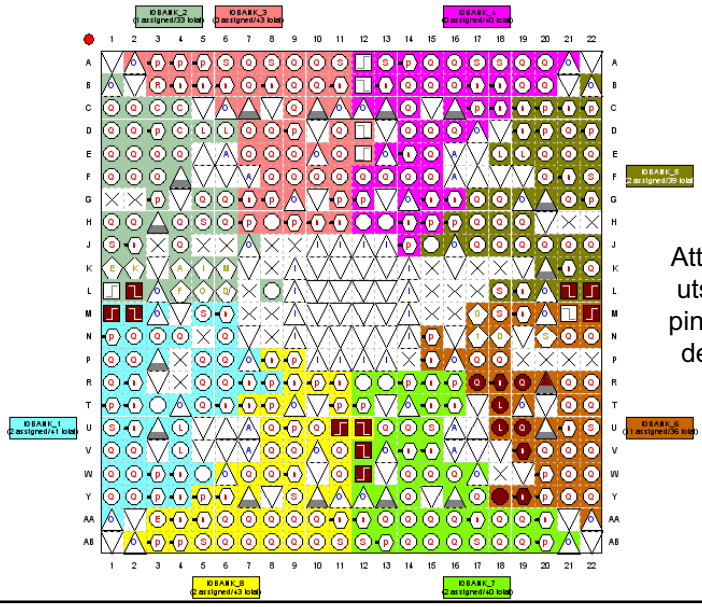
FPGA-kretsens interna layout



Specialiserat klocknät

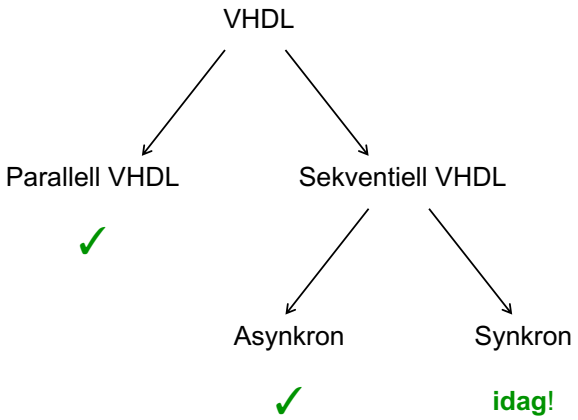


Kretsens pinnar



Att koppla in- och utsignaler till rätt pinnar är en viktig del av syntesen

2. Synkron VHDL



Parallell VHDL:

- I architecture, utanför processer
- All exekveras samtidigt, ordningen är oväsentlig
- Signaltilldelning med <=
- with

Sekventiell VHDL:

- Sker i processer
- Allt exekveras sekventiellt inom processer
- Alla processer exekveras samtidigt
- Variabeltilldelning med :=
- if, case

- Kod som står inom
  `process( ) ... end process`
  är sekventiell VHDL

- Kod som står inom
  `if rising_edge( ) then ... end if`
  eller liknande är synkron (klockad) VHDL

Synkron VHDL är en sorts sekventiell VHDL, där processen aktiveras av klockan

```
process(clk)
begin
     if rising_edge(clk) then
        ...
     end if;
end process;
```

Sensitivitetslista

Trigga på positiv klockflank

Från F2:

- I processens sensitivitetslista ingår de signaler som skall påverka processens igångsättning vid simulering.
- Sensitivitetslistan ignoreras vid syntes

Exempel: shift-register

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

entity shift_register is
port (
  clk: in std_logic;
  x: in std_logic;
  y: out std_logic);
end entity;

architecture arch of shift_register is
  signal r0, r1, r2, r3: std_logic;
begin
  r0 <= x;
  y <= r0 xor r1 xor r2 xor r3;

  process(clk)
  begin
    if rising_edge(clk) then
      r1 <= r0;
      r2 <= r1;
      r3 <= r2;
    end if;
  end process;
end architecture;
```

Vanligt fel:

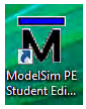
```
if clk='1' then
```

Varför är det fel?

Dessa tre rader innebär inte

```
r3 <= r0
```

Demo



… eller implementerat med en vektor

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

entity shift_register is
port (
  clk: in std_logic;
  x: in std_logic;
  y: out std_logic);
end entity;

architecture arch of shift_register is
  signal r: std_logic_vector(3 downto 0);
begin
  r(0) <= x;
  y <= r(0) xor r(1) xor r(2) xor r(3);

  process(clk)
  begin
    if rising_edge(clk) then
      r(3 downto 1) <= r(2 downto 0);
    end if;
  end process;
end architecture;
```

Shiftregister kan också implementeras med inbyggda operationer `sll`, `srl`, `rol`, `ror`, …

Mer om vektorer

- Om alla bitar skall vara samma kan man skriva

  ```
  f <= ( others => '0')
  ```

  vilket betyder detsamma som `f <= "00000000"`

- Jämförelser kan göras binärt eller som heltal:

  ```
  if x=8 then
  if x="1000" then
  ```

- Vektorer kan i ModelSim kan anges binärt, decimalt eller hexadecimalt:

  ```
  force x 2#1111
  force x 10#15
  force x 16#F
  ```

Heltalsberäkningar

Med tillägget i biblioteket

```
use ieee.std_logic_unsigned.all;
```

kan heltalsberäkningar utföras

Exempel:

```
f <= x + 5;
```

Om `f` och `x` är av typen `std_logic_vector(3 downto 0)` och `x` är `"0001"` (heltalet 1), så blir `f` `"0110"` (heltalet 6)

Klockning av processer

Problem: Antag att en snabb klocka

skall styra en långsam process –

hur implementerar man det?

Lösning: Man skapar en triggsignal genom att dela ned klocksignalen med en räknare

Exempel: räknare

Vi har en 50 MHz klocka (clk50) och vill skapa en långsammare signal:

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

entity timer is
port (
  clk50: in std_logic;
  squarewave: out std_logic);
end entity;

architecture arch of timer is
  signal counter: std_logic_vector(26 downto 0);
begin
  process(clk50)
  begin
    if rising_edge(clk50) then
      counter <= counter + 1;
    end if;
  end process;
  squarewave <= counter(26);
end architecture;
```

Nytt bibliotek

Heltalssummering, börjar om vid 0 när summan går i taket

Demo?
Nja, inte än...

Initialisering av variabler

Problem: Föregående VHDL-kod kan inte simuleras, eftersom `counter` aldrig initialiseras. När strömmen slås på till ett chip, har alla minnesenheter (register) slumpmässigt innehåll.

Lösning: Inför en reset-signal.

Varning: VHDL-syntaxen tillåter att signaler och variabler ges initial-värden, men dessa används bara vid simulering, inte syntes. Undvik!

Räknare med asynkron reset

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

entity timer is
port (
  clk50: in std_logic;
  reset: in std_logic;
  squarewave: out std_logic);
end entity;

architecture arch of timer is
  signal counter: std_logic_vector(26 downto 0);
begin
  process(clk50,reset)
  begin
    if reset='1' then
      counter <= (others => '0');
    elsif rising_edge(clk50) then
      counter <= counter + 1;
    end if;
  end process;
  squarewave <= counter(26);
end architecture;
```

reset-signal

reset-signalen måste ingå i sensitivitetslistan. (Vad händer annars?)

reset-signalen kollas utanför den klockade delen av processen

Räknare med synkron reset

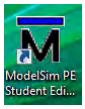
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity timer is
port (
  clk50: in std_logic;
  reset: in std_logic;
  squarewave: out std_logic);
end entity;

architecture arch of timer is
  signal counter: std_logic_vector(3 downto 0);
begin
  process(clk50)
  begin
    if rising_edge(clk50) then
      if reset='1' then
        counter <= (others => '0');
      else
        counter <= counter + 1;
      end if;
    end if;
  end process;
  squarewave <= counter(3);
end architecture;
```

reset-signalen behöver inte ingå i sensitivitetslistan

reset-signalen kollas inuti den klockade delen av processen

lägre värde för att underlätta demon

Demo? OK!



Räknare med andra frekvenser

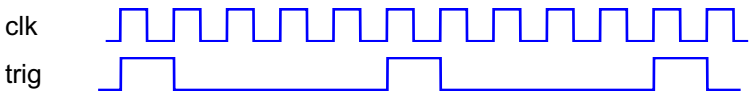
Problem: Föregående VHDL-kod kan endast dela ned frekvenser med en tvåpotens

Lösning: En mer flexibel räknare fås genom att nollställa `counter` när den uppnått ett visst värde

Undvik olikhetsjämförelser, som kostar mycket hårdvara.

Digitala triggsignaler

(kallas även ”aktiveringssignal” eller ”enable”)



1. Skapa ”trig”:

```
process(clk, reset)
begin
  if reset='1' then
    counter <= (others => '0');
  elsif rising_edge(clk) then
    if counter=4 then
      trig <= '1';
      counter <= (others => '0');
    else
      trig <= '0';
      counter <= counter+1;
    end if;
  end if;
end process;
```

2. Använd ”trig”:

```
process(...)
begin
  if rising_edge(clk) then
    if trig='1' then
      ...
    end if;
  end if;
end process;
```

Vilken periodtid?

Svar: 5

Var rädd om klockan

- Synkron elektronik förutsätter att kretsarna klockas exakt samtidigt
- Syntesverktygen implementerar därför klockan separat från andra signaler
- Stör inte klockans funktion!

Exempel på olämplig klockning

- *Misstag 1:* Kombinera inte `rising_edge` med andra villkor.

```
if rising_edge(clk) and (...) then
```

kan införa små fördröjningar som hindrar komponenterna från att klockas samtidigt.

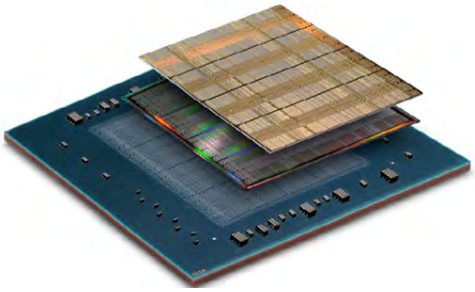
- *Misstag 2:* Använd inte `rising_edge` på användar-definierade signaler.

```
if rising_edge(trig) then
```

där `trig` definierades i bild 24, hindrar syntesverktyget från att använda FPGAns specialiserade klocknät.

3. Labkortet DE1

Altera DE1 Board



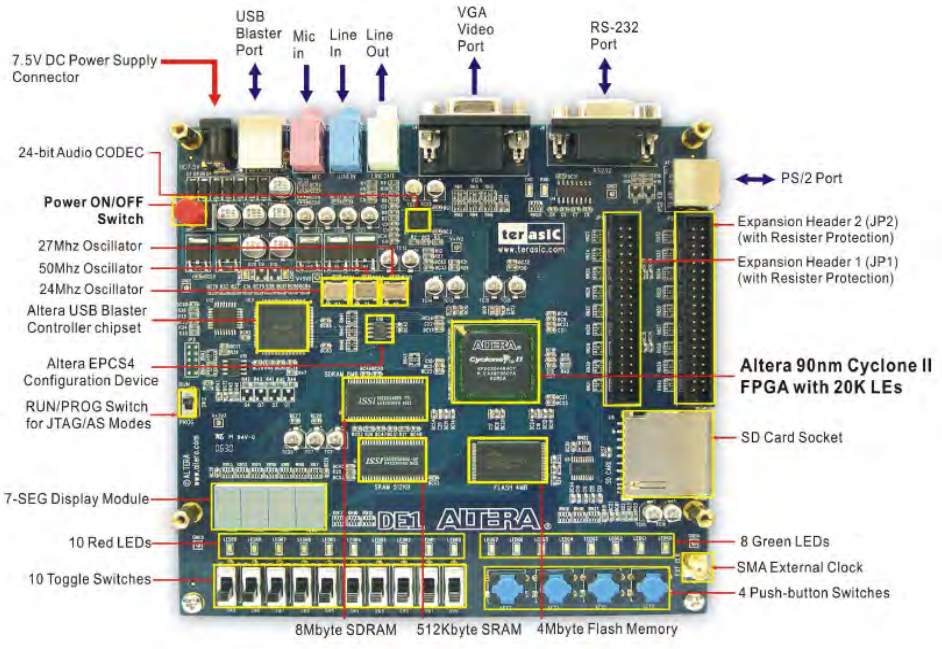
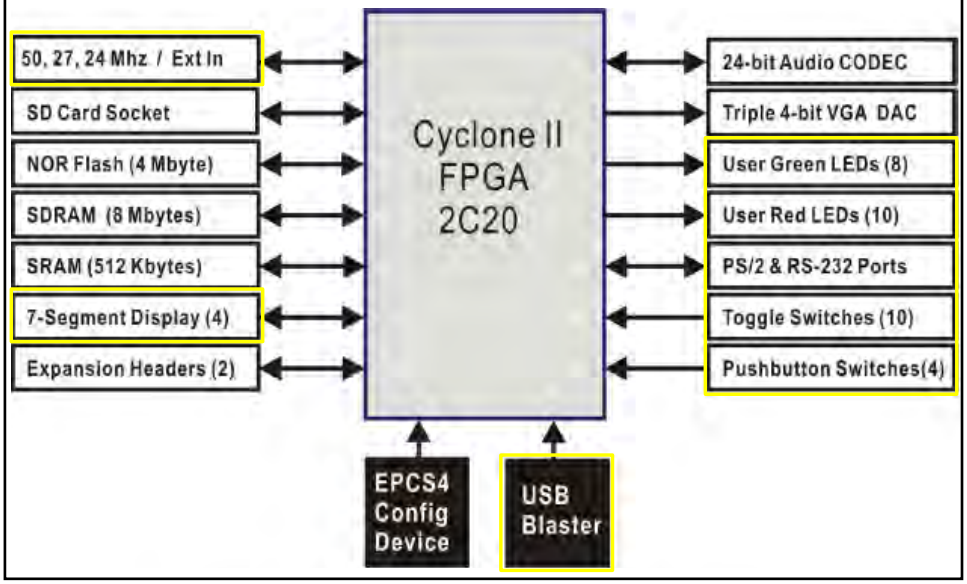


Figure 2.1. The DE1 board.

Input/output



Pushbutton switches

- 4 pushbutton switches
- Debounced by a Schmitt trigger circuit
- Normally high; generates one active-low pulse when the switch is pressed

Toggle switches

- 10 toggle switches for user inputs
- A switch causes logic 0 when in the DOWN (closest to the edge of the DE1 board) position and logic 1 when in the UP position

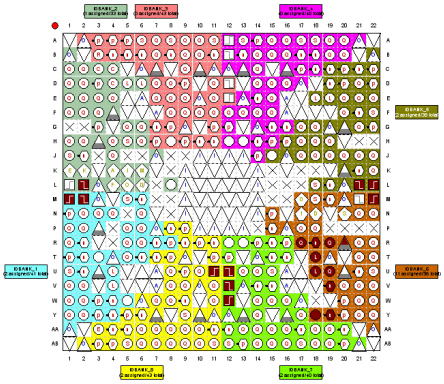
Clock inputs

- 50-MHz oscillator
- 27-MHz oscillator
- 24-MHz oscillator
- SMA external clock input

Pinn-konfigurering

De flesta av FPGA-kretsens pinnar sitter ihop med DE1-kortets kringutrustning:

- omkopplare
- tryckknappar
- LED-arrayer
- 7-segmentsdisplay
- klocksignaler
- …



Vid FPGA-programmering måste man definiera vilka pinnar på kretsen som hör till vilka signaler i FPGA-koden. Det kallas ”pinn-konfigurering”.

Exempel på pinnar

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_R22	Pushbutton[0]
KEY[1]	PIN_R21	Pushbutton[1]
KEY[2]	PIN_T22	Pushbutton[2]
KEY[3]	PIN_T21	Pushbutton[3]
LEDG[0]	PIN_U22	LED Green[0]
LEDG[1]	PIN_U21	LED Green[1]
LEDG[2]	PIN_V22	LED Green[2]
LEDG[3]	PIN_V21	LED Green[3]
LEDG[4]	PIN_W22	LED Green[4]
LEDG[5]	PIN_W21	LED Green[5]
LEDG[6]	PIN_Y22	LED Green[6]
LEDG[7]	PIN_Y21	LED Green[7]
CLOCK_50	PIN_L1	50 MHz clock input

(ur DE1 User Manual, kap 4)

Egna pinn-tabeller

- För återkommande projekt gör man gärna en pinn-tabell i en separat fil
- Hemsidans tabell ”Fixed_DE1_pin_assignments” är en bra utgångspunkt

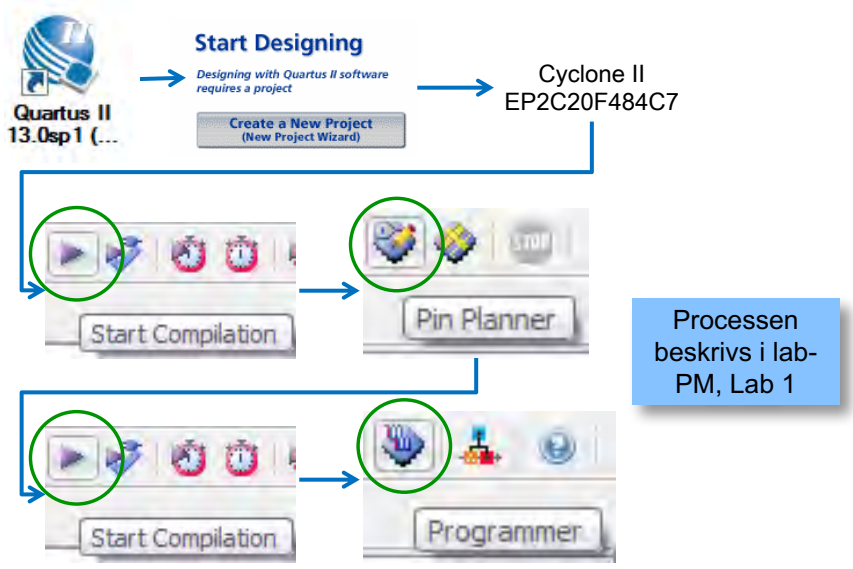
	B	D	E	H
1	Name	Location	Enabled	I/O Standard
2	CLOCK_50	PIN_L1	Yes	LVTTL
3				
4	SW[0]	PIN_L22	Yes	LVTTL
5	SW[1]	PIN_L21	Yes	LVTTL
6	SW[2]	PIN_M22	Yes	LVTTL
7	SW[3]	PIN_V12	Yes	LVTTL
8	SW[4]	PIN_W12	Yes	LVTTL
9	SW[5]	PIN_U12	Yes	LVTTL
10	SW[6]	PIN_U11	Yes	LVTTL
11	SW[7]	PIN_M2	Yes	LVTTL
12	SW[8]	PIN_M1	Yes	LVTTL
13	SW[9]	PIN_L2	Yes	LVTTL
14				
15	KEY[0]	PIN_R22	Yes	LVTTL
16	KEY[1]	PIN_R21	Yes	LVTTL
17	KEY[2]	PIN_T22	Yes	LVTTL
18	KEY[3]	PIN_T21	Yes	LVTTL
19				
20	LEDR[0]	PIN_R20	Yes	LVTTL
21	LEDR[1]	PIN_R19	Yes	LVTTL
22	LEDR[2]	PIN_U19	Yes	LVTTL
23	LEDR[3]	PIN_Y19	Yes	LVTTL
24	LEDR[4]	PIN_T18	Yes	LVTTL
25	LEDR[5]	PIN_V19	Yes	LVTTL
26	LEDR[6]	PIN_Y18	Yes	LVTTL
27	LEDR[7]	PIN_U18	Yes	LVTTL

4. Mjukvaran Quartus

Används för syntes av VHDL, d.v.s.

- kompilera VHDL
- pinn-konfigurering
- tanka ned kod till FPGAn

Arbetsgång



Processen beskrivs i lab-PM, Lab 1

Quartus-varningar

Quartus genererar många varningar vid kompilering!

- 15–20 varningar är normalt även för små, felfria program
- De flesta är ofarliga, t.ex. om timing och kapacitanser
- Några är viktiga, t.ex. om pinntilldelning som saknas och om ”latchar”

Latch = minneselement, D-vippa

Exempel på en ”latch”

Någon ville ha en krets som kopplar omkopplaren SW till lysdioden LED varje gång man trycker på knappen KEY. Vad blev fel?

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

entity switch is
port (
  KEY: in std_logic;
  SW: in std_logic;
  LED: out std_logic);
end entity;

architecture arch of switch is
begin
  process(SW, KEY)
  begin
    if KEY='1' then
      LED <= SW;
    end if;
  end process;
end architecture;
```


Vad Quartus säger…

```
Warning (10631): VHDL Process Statement warning at
switch.vhd(12): inferring latch(es) for signal or
variable "LED", which holds its previous value in
one or more paths through the process
```

…och vad Quartus menar:

Du har otilldelade utsignaler.

Lösningar

Först måste man bestämma sig för vad som skall hända när knappen *inte* är nedtryckt.

Metod 1 (else):

```
process(SW, KEY)
begin
  if KEY='1' then
    LED <= SW;
  else
    LED <= '0';
  end if;
end process;
```

Metod 2 (default-tilldelning):

```
process(SW, KEY)
begin
  LED <= '0'
  if KEY='1' then
    LED <= SW;
  end if;
end process;
```


Implicit minne

Ibland kan man vilja skapa en ”latch” avsiktligt i *synkron VHDL* genom att *inte* täcka alla alternativ. Det kallas då implicit minne.

```
-- T-flipflop: change output
-- value if t=1
library ieee;
use ieee.std_logic_1164.all;

entity Tvippa is port (
  t: in std_logic;
  clk, reset: in std_logic;
  q: out std_logic);
end entity;
```

Om en utsignal i synkron VHDL inte tilldelas ett värde, behålls värdet från förra klockcykeln

```
architecture arch of Tvippa is
  signal s: std_logic;
begin
  process(clk,reset)
  begin
    if reset='1' then
      s <= '0';
    elsif rising_edge(clk) then
      if t='1' then
        s <= not s;
      end if;
    end if;
  end process;
  q <= s;
end architecture;
```

Veckosammanfattning

Efter den här veckan behärskar ni…

- VHDLs grunder
  - bibliotek, entitet och arkitektur
  - in- och ut-signaler
  - signaltilldelning med **<=**
  - vektorer
  - heltalsaddition
- Parallell VHDL (utanför process)
  - grindnivå (**and**, **or**, **not**)
  - interna signaler
  - **with**

- Sekventiell VHDL (innanför process)
  - sensitivitetslista
  - variabler
  - **if** och **case**
  - asynkron VHDL (utanför **if rising_edge( )**)
  - synkron VHDL (innanför **if rising_edge( )**)
  - reset-signaler (asynkron och synkron)
- Simulering med ModelSim

Labinformation

- Anmäl er till labgrupper (de som inte redan gjort det)
- De som anmält sig ensamma får gärna bilda par
- Lab-PM för lab 1 finns på hemsidan
- Obligatoriska förberedelseuppgifter före varje lab
- Man måste ha löst alla förberedelseuppgifterna före labben, men det måste inte vara 100% rätt.
- Båda personerna i gruppen skall kunna redovisa förberedelseuppgifterna.

Läs & lös

Läs:

- Kompendiet ”Kretskonstruktion med VHDL”, avsnitt 4–5 och 10
- Lab-PM för lab 1

Lös:

- Förberedelseuppgifter till lab 1 (måste vara gjorda före labben)

Föreläsning 4
D/A-omvandling

Erik Agrell 2017-09-05

Innehåll:

1. Tre sätt att simulera i ModelSim
2. Principer för D/A- och A/D-omvandling
3. Konstruktion av D/A-omvandlare
4. Bipolär D/A-omvandling

Bilder av E. Agrell och A. Linde

Veckosammanfattning

Efter förra veckan behärskar ni…

- VHDLs grunder
  - bibliotek, entitet och arkitektur
  - in- och ut-signaler
  - signaltilldelning med **<=**
  - vektorer
  - heltalsaddition
- Parallell VHDL (utanför process)
  - grindnivå (**and**, **or**, **not**)
  - interna signaler
  - **with**

- Sekventiell VHDL (innanför process)
  - sensitivitetslista
  - variabler
  - **if** och **case**
  - asynkron VHDL (utanför **if rising_edge( )**)
  - synkron VHDL (innanför **if rising_edge( )**)
  - reset-signaler (asynkron och synkron)
- Simulering med ModelSim

1. Tre sätt att simulera i ModelSim

1. Klicka och skriv kommandon
   - som vi har gjort hittills (Add Wave, force, run,…)
2. Exekvera en do-fil (script)
   - praktiskt om man vill köra samma simulering flera gånger, t.ex. vid felsökning
3. Kör ett VHDL-program för att testa ett annat (testbänk)
   - inte i den här kursen, men viktigt i stora projekt

Do-filer

Arbetsgång:

1. Starta ModelSim och öppna ett projekt med en VHDL-fil, t.ex. and-grinden från föreläsning 1 och 2:

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

entity and_gate is
port (
  x,y: in std_logic;
  f: out std_logic);
end entity;

architecture arch of and_gate is
begin
  f <= (x and y);
end architecture;
```

2. Kompilera

3. Välj *File* > *New* > *Source* > *Do*
4. Skriv eller klistra in lämpliga ModelSim-kommandon, t.ex.

```
# standard header:
vsim work.and_gate
view wave
add wave *

# simulation commands:
force x 0 0ns, 1 200ns
force y 0 0ns, 1 100ns, 0 400ns
run 500ns
```

*Starta simulering och välj entitet **and_gate***

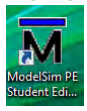
*Lägg till alla signaler till **wave**-fönstret*

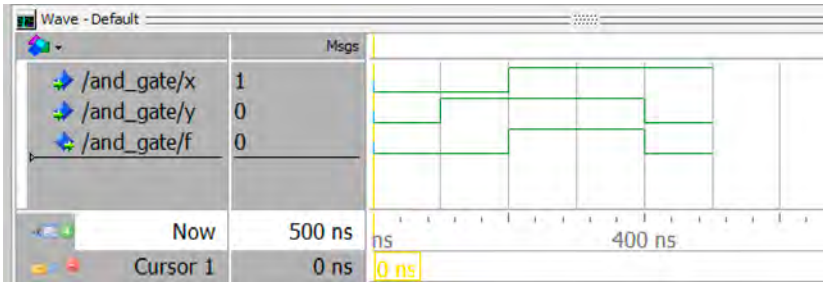
ModelSims simulerings-kommandon

- De tre första raderna är ett standardhuvud som man kan använda i alla sina do-filer (och ändra entitetsnamnet på första raden)
- Kommentarer markeras med #
- Simuleringskommandon kan kombineras med semikolon

5. Spara filen och kalla den t.ex. **and_gate.do**
6. Ekekvera do-filen genom att skriva
   `do and_gate.do`
   i kommandofönstret

Demo





Samma resultat som när vi simulerade manuellt

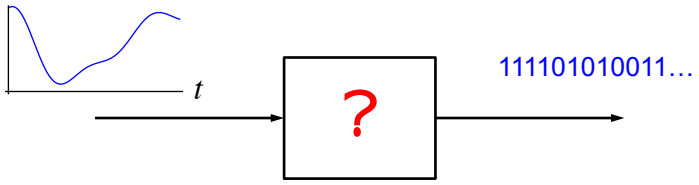
Textfiler

VHDL-filer (.vhd) och DO-filer (.do)
sparas som ren text

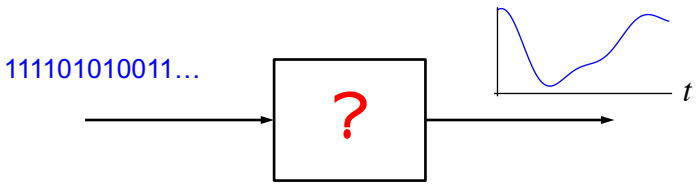
Kan redigeras i vilken editor som helst,
även utanför ModelSim och Quartus

2. Principer för D/A- och A/D-omvandling

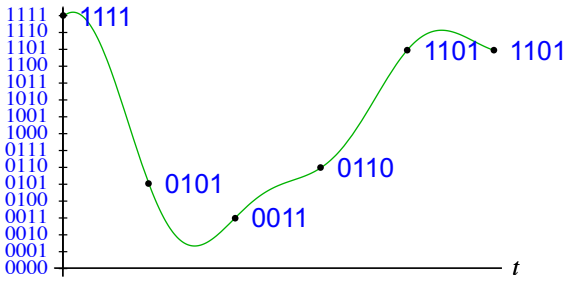
Ofta har man en vågform (ljud, video, mätdata,…) som man vill representera som bitar:



Eller tvärt om:



A/D: Från vågform till bitar

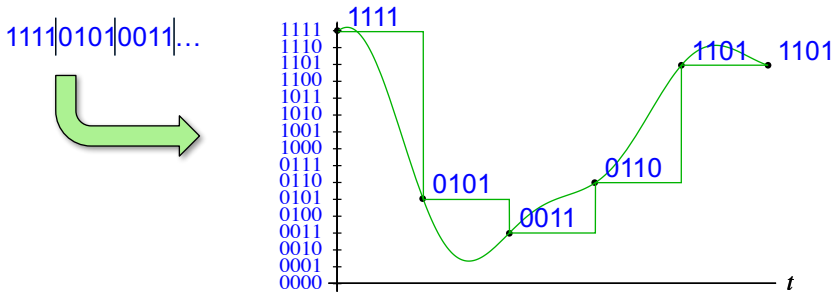


1. Sampling:
   Hacka signalen i tiden

2. A/D-omvandling (kvantisering):
   Representera varje tidssampel som en sträng bitar

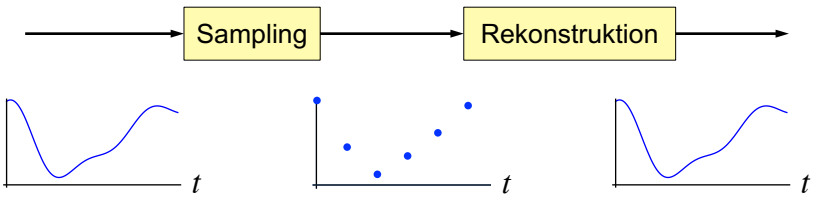
3. Sätt ihop strängarna

D/A: Från bitar till vågform



1. Dela upp bitsekvensen i ord av samma längd
2. D/A-omvandling:
   Representera varje digitalt ord som en spänningsnivå (eller ström)
3. Låt utspänningen ligga konstant tills nästa sampel kommer
4. Lågpassfiltrera

Samplingsteoremet



Villkor för att signalen skall kunna rekonstrueras utan distortion:

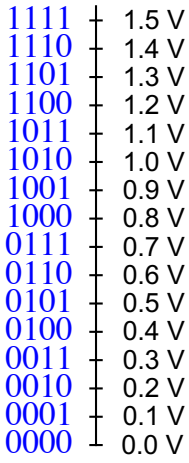
$$f_s \geq 2 f_{max}$$

f_s: samplingsfrekvensen
f_{max}: signalens högsta frekvens

3. Konstruktion av D/A-omvandlare

Digital/analog (D/A)-omvandling:
att omvandla bitar till spänning
eller ström

D/A-omvandlare finns som
integrerade kretsar, t.ex. DAC0808



Datablad

Att läsa datablad är ofta en förutsättning för att använda komponenten

National Semiconductor

May 1999

DAC0808
8-Bit D/A Converter

DAC0808 8-Bit D/A Converter

General Description

The DAC0808 is an 8-bit monolithic digital-to-analog converter (DAC) featuring a full scale output current settling time of 150 ns while dissipating only 33 mW with ±5V supplies. No reference current (I_{REF}) trimming is required for most applications since the full scale output current is typically ±1 LSB of 255 I_{REF}/256. Relative accuracies of better than ±0.19% assure 8-bit monotonicity and linearity while zero level output current of less than 4 µA provides 8-bit zero accuracy for I_{REF}≥2 mA. The power supply currents of the DAC0808 is independent of bit codes, and exhibits essentially constant device characteristics over the entire supply voltage range.

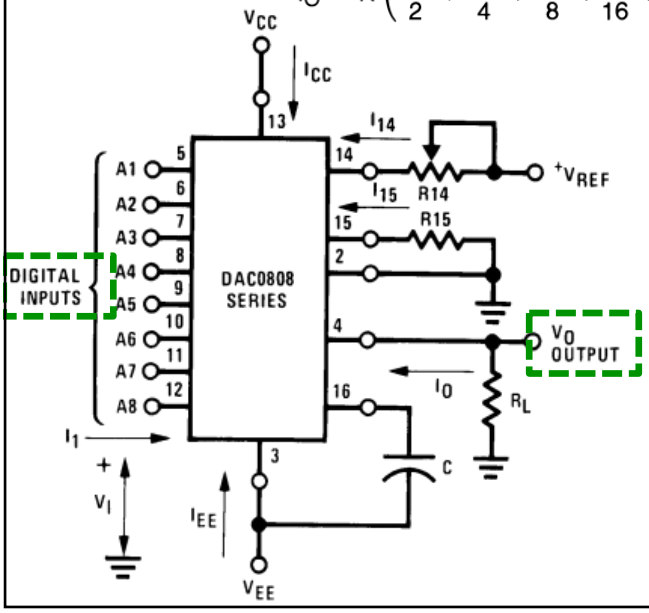
The DAC0808 will interface directly with popular TTL, DTL or CMOS logic levels, and is a direct replacement for the MC1508/MC1408. For higher speed applications, see DAC0800 data sheet.

Features

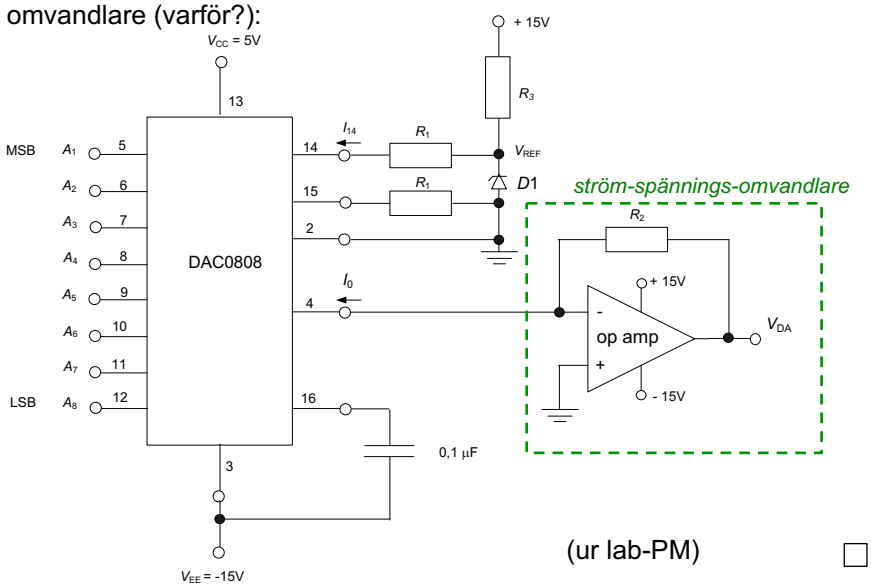
- Relative accuracy: ±0.19% error maximum
- Full scale current match: ±1 LSB typ
- Fast settling time: 150 ns typ
- Noninverting digital inputs are TTL and CMOS compatible
- High speed multiplying input slew rate: 8 mA/µs
- Power supply voltage range: ±4.5V to ±18V
- Low power consumption: 33 mW @ ±5V

$$I_O = K\left(\frac{A1}{2} + \frac{A2}{4} + \frac{A3}{8} + \frac{A4}{16} + \frac{A5}{32} + \frac{A6}{64} + \frac{A7}{128} + \frac{A8}{256}\right)$$

where $K \cong \frac{V_{REF}}{R14}$



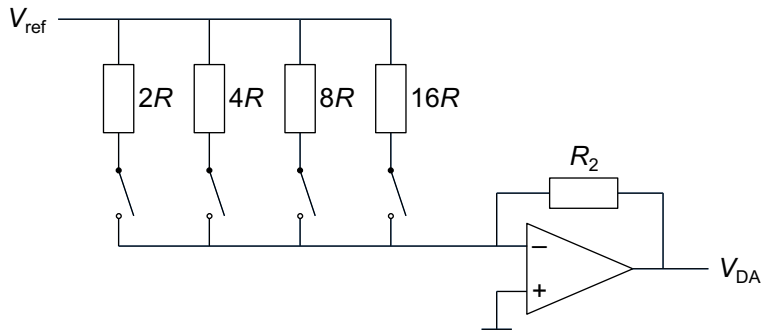
Vi kombinerar D/A-omvandlaren med en ström-spännings-omvandlare (varför?):



(ur lab-PM)

Två konstruktionsmetoder av D/A

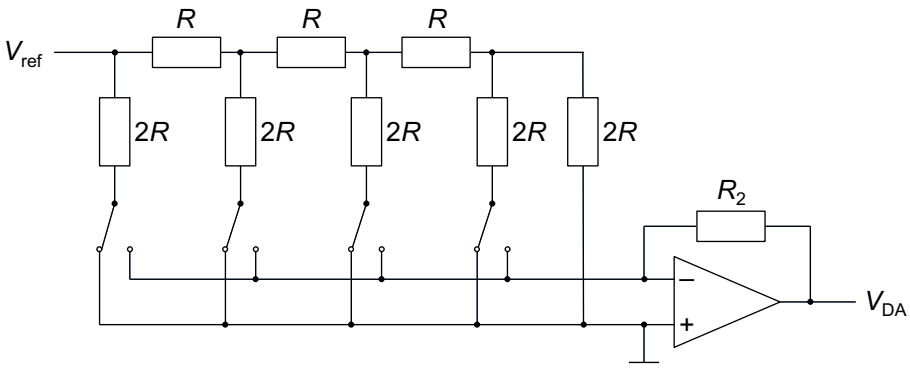
1. Viktade resistanser



Enkel men onoggrann konstruktion

(varför?)

2. R-2R-stege



Stabil och populär konstruktion

(varför?)

Varför R-2R-stege?

- Alltid samma ström genom motstånden, oavsett omkopplarnas lägen
- I kretsen behöver endast två storlekar på motstånd implementeras (eller en storlek, om $2R$ realiseras som $R+R$ i serie)
- I integrerad kretsteknologi kan motstånd *av samma storlek* konstrueras mycket exakt

⇒ Hög noggrannhet, även vid hög ordlängd (många bitar)

Ur databladet för DAC0808

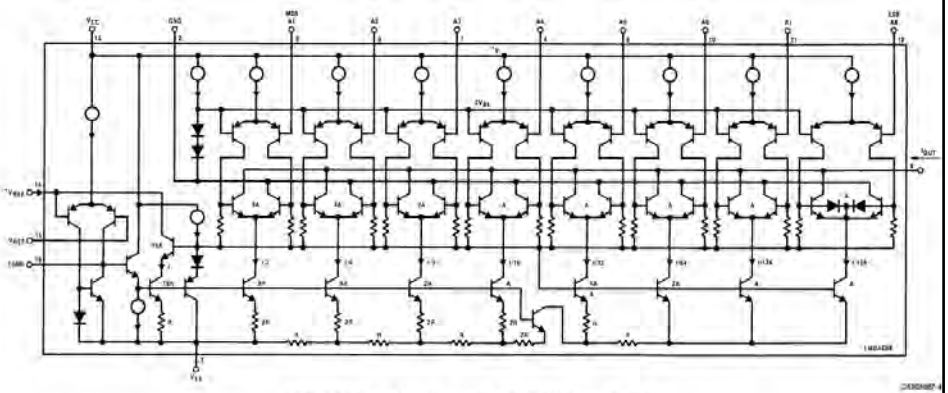
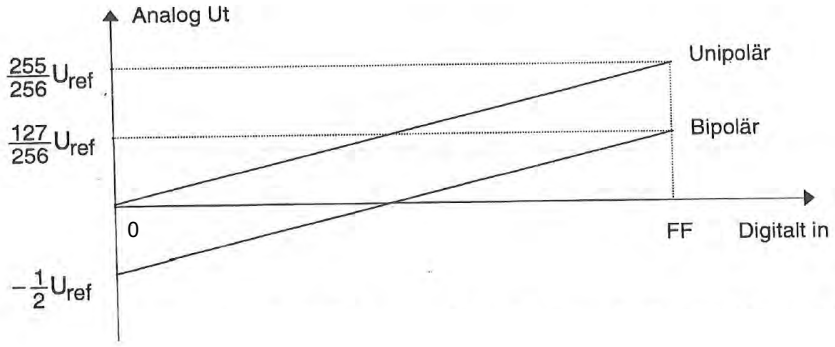


FIGURE 2. Equivalent Circuit of the DAC0808 Series (Note 8)

4. Bipolär D/A-omvandling

Spänningsomfånget för en D/A-omvandlare kan vara mellan 0 och U_{ref} (unipolär) eller mellan $\pm U_{ref}/2$



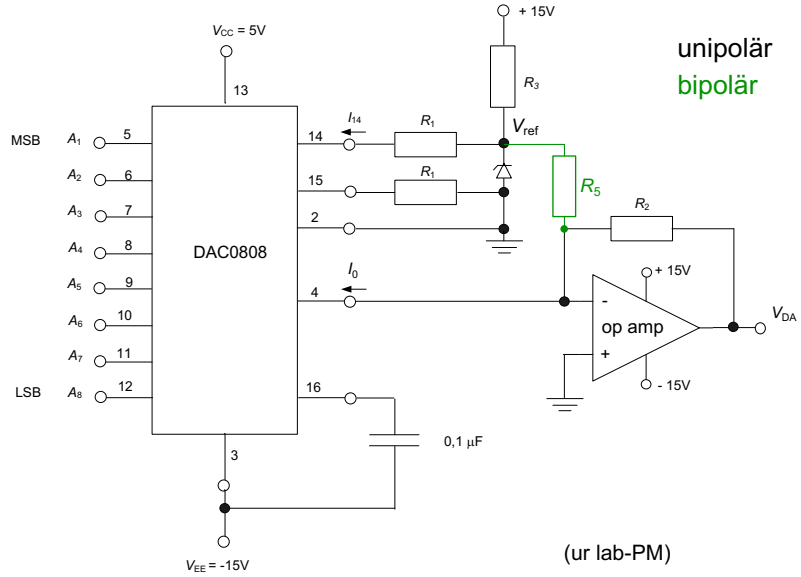
Unipolär (från bild 13):

1111	1.5 V
1110	1.4 V
1101	1.3 V
1100	1.2 V
1011	1.1 V
1010	1.0 V
1001	0.9 V
1000	0.8 V
0111	0.7 V
0110	0.6 V
0101	0.5 V
0100	0.4 V
0011	0.3 V
0010	0.2 V
0001	0.1 V
0000	0.0 V

Bipolär:

1111	0.7 V
1110	0.6 V
1101	0.5 V
1100	0.4 V
1011	0.3 V
1010	0.2 V
1001	0.1 V
1000	0.0 V
0111	–0.1 V
0110	–0.2 V
0101	–0.3 V
0100	–0.4 V
0011	–0.5 V
0010	–0.6 V
0001	–0.7 V
0000	–0.8 V

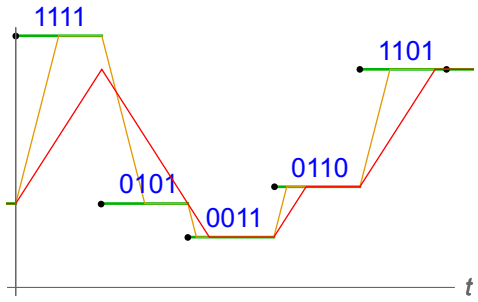
Modifiering av D/A-kopplingen



(ur lab-PM)

- Samma integrerade krets kan användas för unipolär som bipolär D/A
- Nollpunkten kan förskjutas med en ström-bias på kretsens utgång
- Kopplingen på förra sidan blir bipolär om strömmen genom R_5 är $I_{0max}/2$ (förklara varför!)

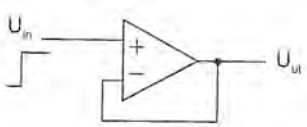
D/A-omvandlarens tidsegenskaper



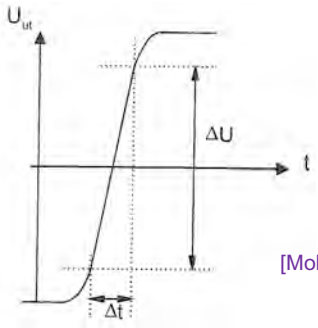
- Grönt: Ideal utsignal, stegfunktion
- Gult: Mer realistisk utsignal, OK efter filtrering
- Rött: För långsam utsignal, ger allvarlig distortion

Slew rate

Slew rate = maximal spänningsförändring per tid



$$\text{Slew rate} = \frac{\Delta U}{\Delta t}$$



[Molin s 74]

Slew rate påverkar hur snabbt varierande signaler elektroniken kan hantera, t.ex. hur snabbt D/A-omvandlaren kan klockas

Läs & lös

Läs:

- Läs Molin 1.3: Analogt kontra digitalt
- Läs Bengtsson 4.2: D/A-omvandling
- Läs (översiktligt) databladet DAC0808
- (Vid behov, repetera Molin 2.3: Op-förstärkarkopplingar)

Lös:

- I inlämningsuppgift 1 lämnas displayen blank om alla switcharna står på 1

Föreläsning 5
LTspice

Erik Agrell 2017-09-06

Innehåll:

1. SPICE översikt
2. Rita kopplingsschema i LTspice
3. Simulering i LTspice

Bilder av G. Hult och E. Agrell

1. SPICE översikt

- Simulation Program with Integrated Circuit Emphasis (SPICE)
- Utvecklades 1973 på University of California, Berkeley
- Ursprungligen textbaserat, grafiskt gränssnitt 1989
- Används globalt, i både utbildning och utveckling
- Öppen källkod
- Analyserar kretsar med hjälp av nodanalys
- Populära kloner: Pspice, Xspice, Hspice och LTspice

LTspice

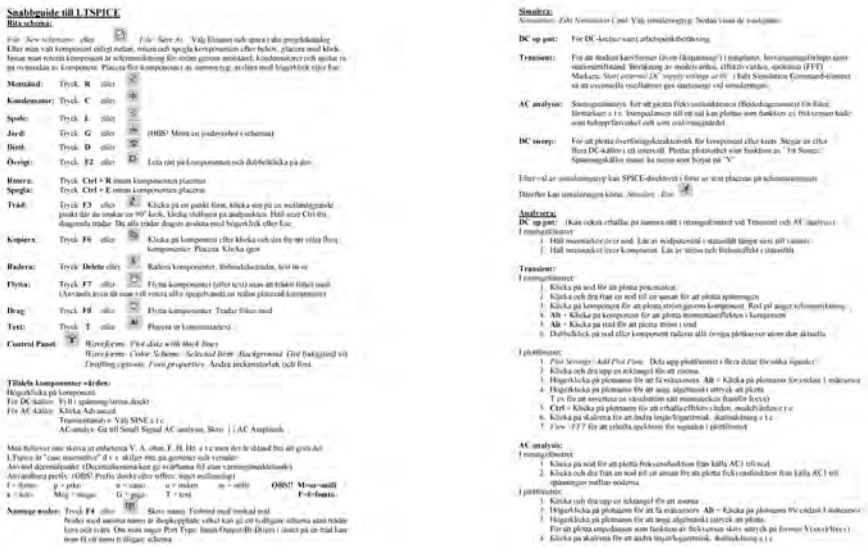
LTspice produceras av Linear Technology Corp (LTC) och innehåller modeller för många av LTCs egna kretsar

- Gratis på linear.com/ltspice
- Finns i datasalarna
- Finns i labben
- Mac-versionen av LTspice rekommenderas inte 

Arbetsgång

1. Nytt kopplingsschema
2. Rita kretsen
   - Placera komponenter
   - Koppla ihop komponenterna
   - Ange komponentvärden
3. Simulera
4. Plotta resultat

Görans snabbguide



2. Rita kopplingsschema i LTspice

Rotera: Tryck **Ctrl + R** innan komponenten placeras

Spegla: Tryck **Ctrl + E** innan komponenten placeras

Tråd: Tryck **F3** eller  Klicka på en punkt först, klicka sen på en mellanliggande punkt där du önskar en 90° krök, klicka slutligen på ändpunkten. Håll nere Ctrl för diagonala trådar. Då alla trådar dragits avsluta med högerklick eller Esc.

Kopiera: Tryck **F6** eller  Klicka på komponent eller klicka och dra för att välja flera komponenter. Placera. Klicka igen

Radera: Tryck **Delete** eller  Radera komponenter, förbindelsetrådar, text m m.

Flytta: Tryck **F7** eller  Flytta komponenter (eller text) utan att trådar följer med. (Används även då man vill rotera eller spegelvända en redan placerad komponent)

Drag: Tryck **F8** eller  Flytta komponenter. Trådar följer med.

Komponenter

Motstånd: Tryck **R** eller

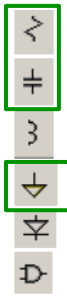
Kondensator: Tryck **C** eller

Spole: Tryck **L** eller

Jord: Tryck **G** eller (OBS! Minst en jordsymbol i schemat)

Diod: Tryck **D** eller

Övrigt: Tryck **F2** eller Leta rätt på komponenten och dubbelklicka på den.



voltage	current
npn	pnp
nmos	pmos
zener	
[OpAmps]	
…	

Tilldela komponentvärden

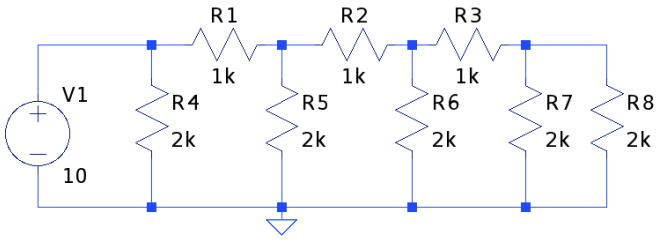
Högerklicka på komponent.
För DC-källor: Fyll i spänning/ström direkt
För AC-källor: Klicka Advanced.
Transientanalys: Välj SINE e t c.
AC-analys: Gå till Small Signal AC analysis, Skriv 1 i AC Amplitude

Man behöver inte skriva ut enheterna V, A, ohm, F, H, Hz e t c men det är ibland bra att göra det.
LTspice är "case insensitive" d v s skiljer inte på gemener och versaler.
Använd decimalpunkt. (Decimalkomma kan ge svårfunna fel utan varningsmeddelande)
Användbara prefix: (OBS! Prefix direkt efter siffror, inget mellanslag)

f = femto	p = piko	n = nano	u = mikro	m = milli
k = kilo	Meg = mega	G = giga	T = tera	

OBS!! M=m=milli
F=f=femto

Exempel 1: motståndsnät



- Komma igång
- Rita kopplingsschema
- Ange komponentvärden

Demo

3. Simulering i LTspice

Simulation>Edit Simulation Cmd. Välj simuleringtyp. Nedan visas de vanligaste:

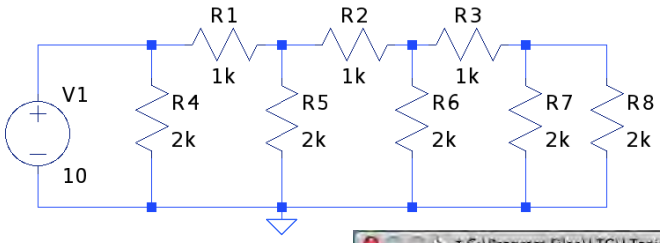
DC op pnt: För DC-kretsar samt arbetspunktberäkning.

Transient: För att studera kurvformer (även likspänning!) i tidsplanet. Insvängningsförlopp samt stationärtillstånd. Beräkning av medelvärden, effektivvärden, spektrum (FFT) Markera: *Start external DC supply voltage at 0V* i Edit Simulation Command-fönstret så att eventuella oscillatorer ges startenergi vid simuleringen.

AC analysis: Småsignalanalys. För att plotta frekvensfunktionen (Bodediagrammet) för filter, förstärkare e t c. Inimpedansen till ett nät kan plottas som funktion av frekvensen både som belopp/fasvinkel och som real/imaginärdel.

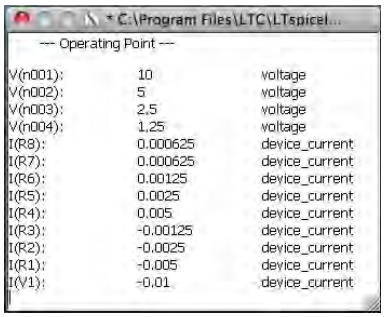
DC sweep: För att plotta överföringskarakteristik för komponent eller krets. Stegar en eller flera DC-källor i ett intervall. Plottar plotstorhet som funktion av ”1st Source” Spänningskällor måste ha namn som börjar på ”V”

Exempel 1 igen



- Beräkna likspänningar och strömmar (”DC op pnt”)
- Plotta tidssignaler (”Transient”)

Demo



Namnge noder

Tryck **F4** eller  Skriv namn. Förbind med önskad nod.

Noder med samma namn är ihopkopplade vilket kan ge ett tydligare schema utan trådar kors och tvärs. Om man anger Port Type: Input/Output/Bi-Direct i slutet på en tråd kan man få ett ännu tydligare schema.

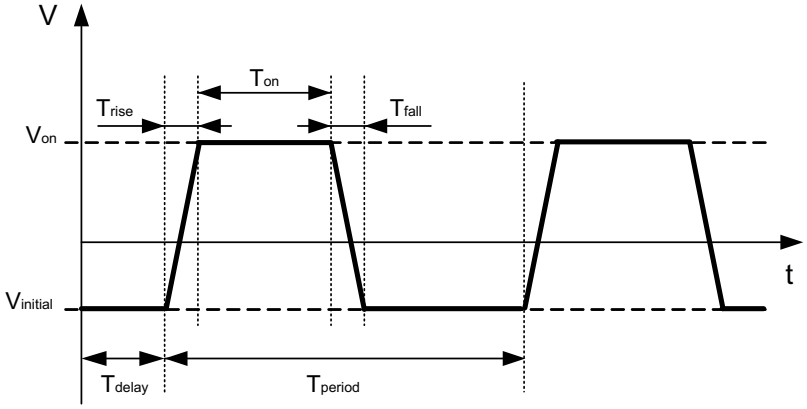
Importerade komponenter

Man kan importera komponenter som inte finns i standardbiblioteket, som man får från andra leverantörer eller definierar själv.

1. Lägg en komponentfil (filtyp *.lib*, t.ex. *extras.lib*) i samma katalog som ditt kopplingsschema (filtyp *.asc*).
2. Lägg symbolfiler (filtyp *.asy*) i samma katalog.
3. Klicka på .op och skriv kommandot *.include* filnamn*.lib*

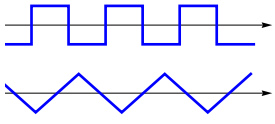
En komponentfil (filnamn*.lib*) kan innehålla flera komponentbeskrivningar.

PULSE-vågens parametrar

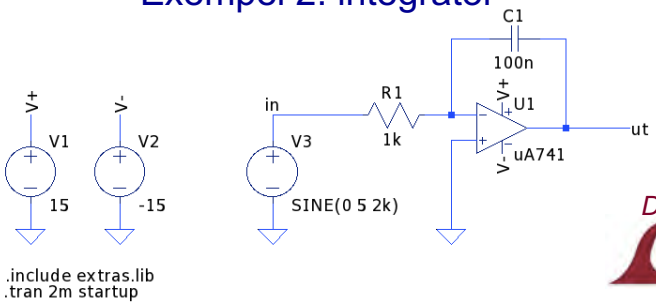


Specialfall:

- Fyrkantvåg: Trise = Tfall = 1p (eller något annat litet värde, men **inte 0**!)
- Triangelvåg: Ton = 0, Tperiod = Trise+Tfall

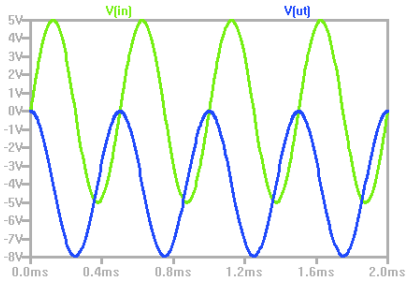


Exempel 2: integrator

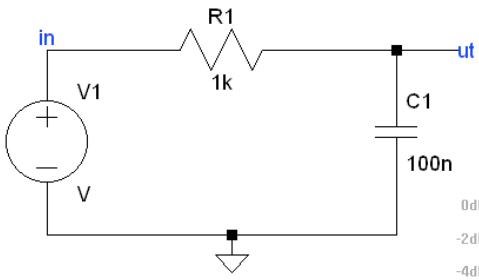


Demo

- Namnge noder (varför?)
- Importera komponenter
- Definiera växelspänningar och fyrkantvåg

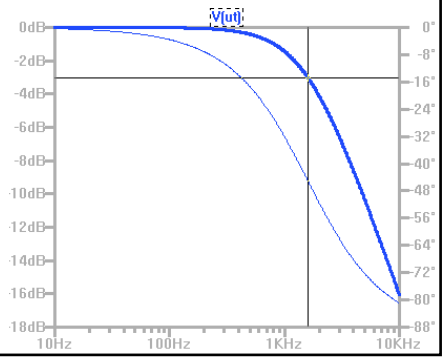


Exempel 3: lågpassfilter



- Bode-diagram (”AC analysis”)
- Avläsning med mätkorset

Demo



Problem med mjukvaran?

Installationsproblem? Felmeddelanden? Konstiga resultat?

Datorövning fredag 8:15–10:00 i Gnistan (F6)

- Tillfälle att fråga
- Ingen genomgång av nytt material
- ModelSim eller LTspice, välj själv
- Förslag på övningsuppgifter kommer att finnas på kurshemsidan

Labschema

- För närvarande har vi många fler anmälda labgrupper på onsdagar än tisdagar
- Fler grupper innebär längre väntan på hjälp

Kan någon labgrupp tänka sig att byta från onsdag till tisdag?

Läs & lös

Läs:

- Läs Göran Hult: ”Snabbguide till LTSPICE” noggrant (2 sidor)
- Skumma igenom Göran Hult: ”LTSPICE: Introduktion till kretssimulering”

Lös:

- Kör igång LTspice och testa några exempel
- Inlämningsuppgift 1 (om ni inte redan har gjort det)
- Förberedelseuppgifter till lab 2

Föreläsning 7
A/D-omvandling och tillståndsmaskiner

Erik Agrell 2017-09-12

Innehåll:

1. Tre sorters A/D-omvandlare
2. Tillståndsmaskiner
3. Digital implementation av SAR

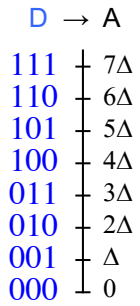
Bilder av E. Agrell och A. Linde

Veckosammanfattning

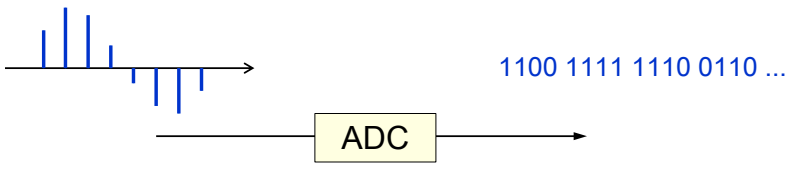
Efter kursvecka 2 behärskar ni…

- VHDL-simulering med ModelSim
  - med kommandon (FORCE, RUN)
  - med do-fil
- VHDL-syntes i Quartus
  - Kompilering
  - Pinn-tilldelning
  - Syntes
  - Test och felsökning

- D/A-omvandling
  - Teoretiska principer
  - Inkoppling av DAC0808
  - Viktade resistanser (inte bra)
  - R-2R-stege (bra)
  - Unipolär och bipolär
- Kretssimulering med LTSpice
  - Rita kopplingsschema
  - Ange komponentvärden och signalkällor
  - Simulera DC_nivåer (Op pnt), tidssignaler (Transient) och frekvensgång (AC analysis)
  - Använda mätkorset



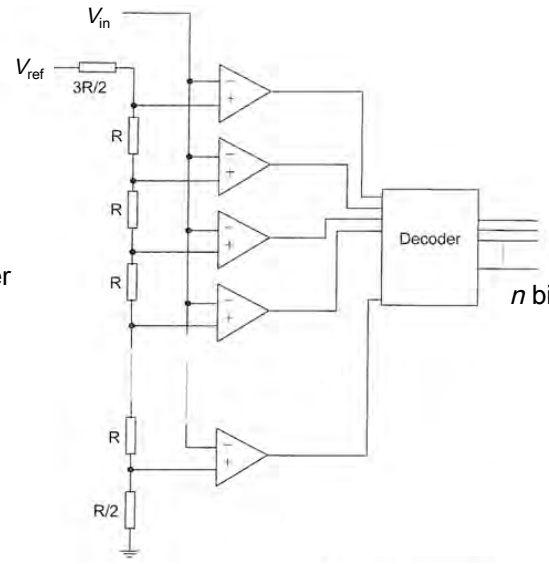
1. Tre sorters A/D-omvandlare



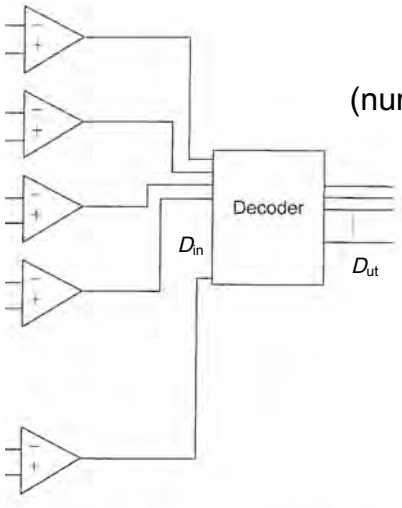
1. Flash-omvandling
2. Successive approximation, SAR
3. Dual-slope-omvandling

Metod 1: Flash-omvandling

2^n motstånd
2^n–1 komparatorer



Flash-omvandlarens avkodare



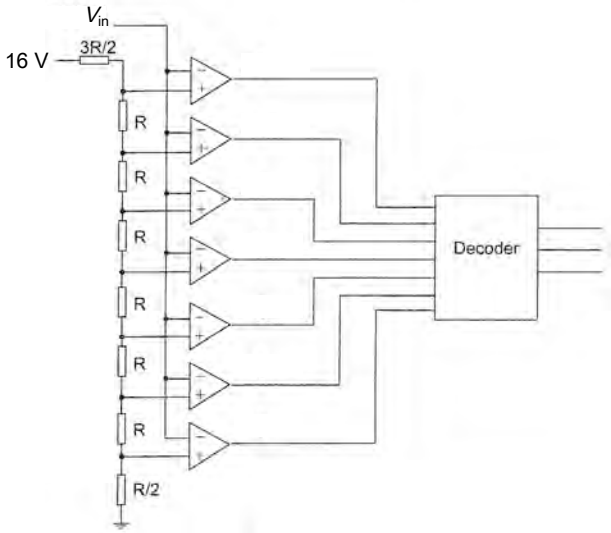
2^n–1 bitar in (numrerade uppifrån)

n bitar ut

D_{in}	D_{ut}
000…00	1…11
100…00	1…10
110…00	1…01
⋮ ⋮	⋮ ⋮
111…10	0…01
111…11	0…00

Exempel

(n=3 bitar)



□

Flashomvandlarens egenskaper:

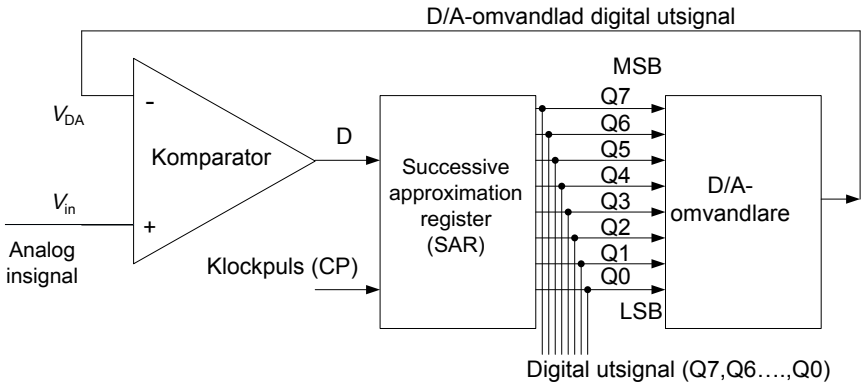
\+ Mycket snabb (konstant tid oavsett n)

– Stor krets (yta ~ 2^n) ⇒ stor kostnad och energiförbrukning

Används i oscilloskop (där snabbhet är viktigare än upplösning)

Metod 2: Successiv approximation

Viktig metod



Varje fråga…

… ger 1 bits information

… halverar osäkerheten

… kostar 1 clockcykel

n frågor…

… ger *n* bits information

… ger upplösningen $\Delta = V_{max}/2^n$

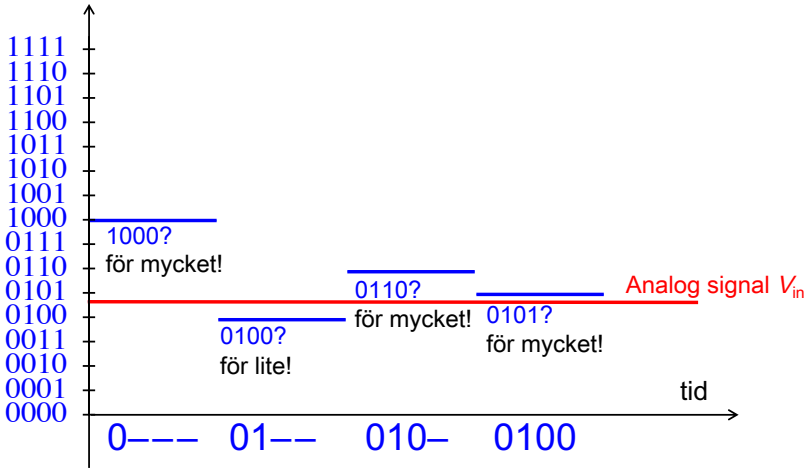
… kostar *n* klockcykler

SAR-algoritmen

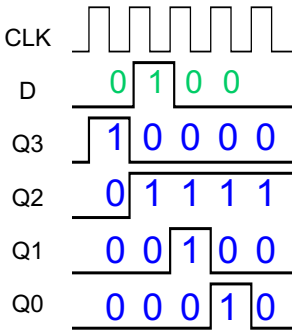
1. Gissa på $V_{max}/2$ — 10000…
2. För lite ⇒ öka med $V_{max}/4$ — 11000…
   För mycket ⇒ minska med $V_{max}/4$ — 01000…
3. För lite ⇒ öka med $V_{max}/8$
   För mycket ⇒ minska med $V_{max}/8$

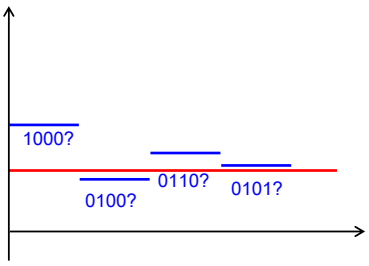
…

Exempel



Tidsdiagram





Efter avslutad omvandling har D-signalen (seriell) fyllt vektorn Q (parallell)

SAR-omvandlarens egenskaper:

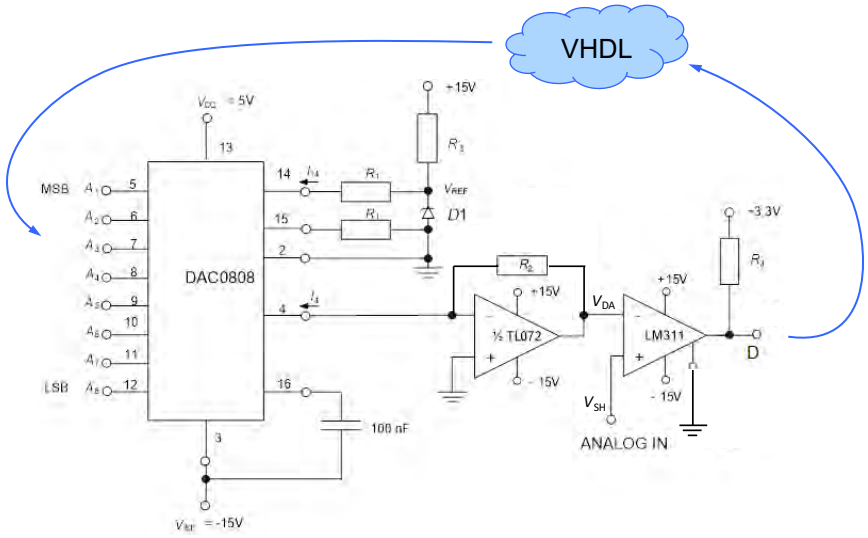
\+ Mycket mindre komplex än flash (yta ~ *n*)

– Långsammare än flash (tid ~ *n*)

– Kräver DAC

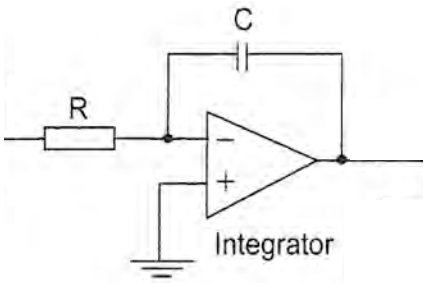
Används i digital kommunikation, t.ex. lab 3

SAR-omvandlaren i lab 3



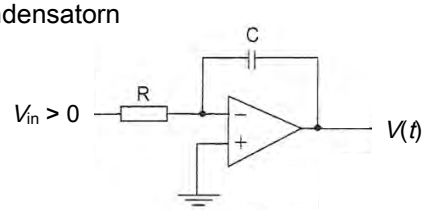
Metod 3: Dual-slope-omvandling

Dual-slope-omvandlingen utnyttjar en integrator, som först laddas upp och sedan laddas ur



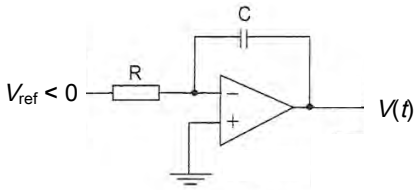
Steg 1: Uppladdning av kondensatorn

$0 \le t < t_1$



Steg 2: Urladdning av kondensatorn

$t_1 \le t < t_1+t_2$

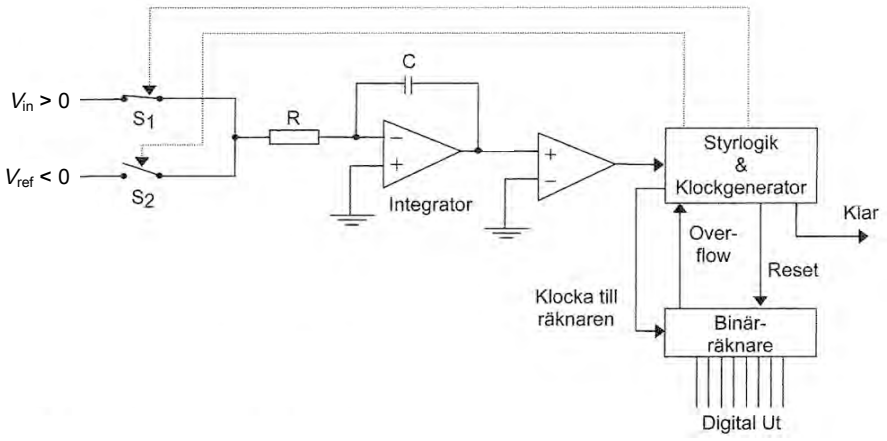


t_2 väljs så att $V(t_1+t_2) = 0$, d.v.s. helt urladdad kondensator

Dual-slope-algoritmen

1. Uppladdning av kondensatorn
   - Okänd spänning V_{in}
   - Känd tid t_1
2. Urladdning av kondensatorn
   - Känd spänning V_{ref}
   - Uppmätt tid t_2
3. Beräkning
   - $V_{in} = -\frac{t_2}{t_1} V_{ref}$

Dual-slope-omvandlarens implementering



Dual-slope-omvandlarens egenskaper:

+ Mycket noggrann
  - Tidmätning kan göras mycket noggrann (kristallklocka)
  - Resultatet är oberoende av *RC*
+ Kräver ingen DAC
– Mycket långsam

Används i voltmetrar (där upplösning är viktigare än snabbhet)

2. Tillståndsmaskiner

- Ett tillståndsmaskin består av ett antal tillstånd (lägen, moder) med övergångar mellan
- Används för att beskriva system vars beteende ändras som en följd av tidigare åtgärder
- Abstrakt? Nej, vi använder tillståndsmaskiner varje dag!
- *Exempel:* Funktionen hos en vanlig dörr

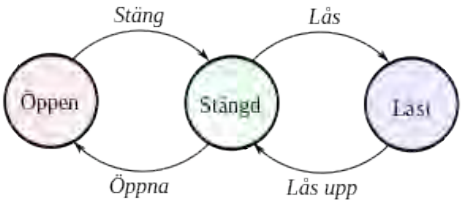
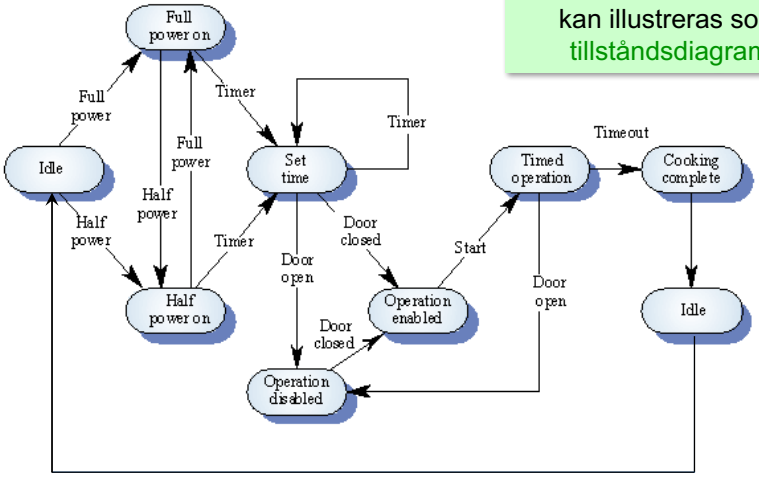


Illustration från Wikpedia

Exempel
(mikrovågsugn)

Tillståndsmaskiner kan illustreras som tillståndsdiagram

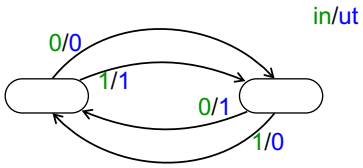


Källa: http://nas.uhcl.edu/helm/swen5231/Ch_16_1/sld017.htm

Digitalt exempel

Invertera varannan bit i en sekvens:

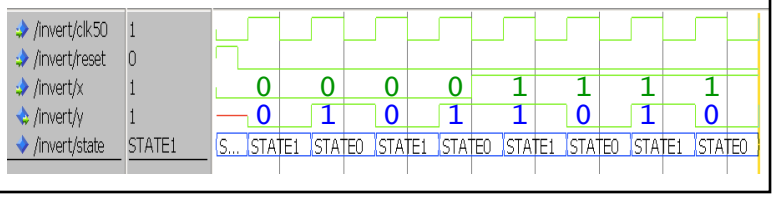
in: 00001111
ut: 01011010



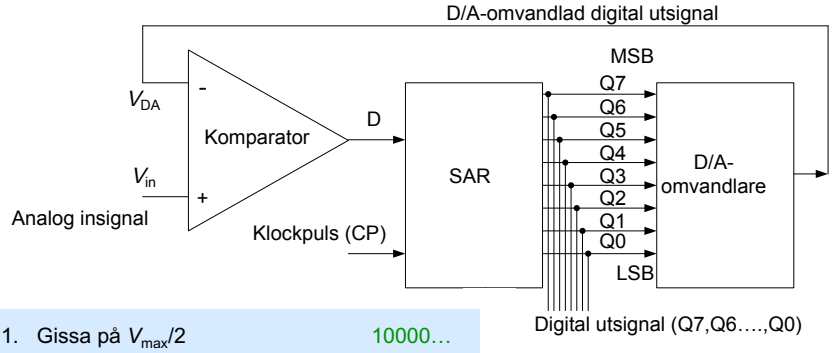
VHDL-kod

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

entity invert is port(
  clk50, reset: in std_logic;
  x: in std_logic;
  y: out std_logic);
end entity;

architecture state_machine of invert is
  type StateType is (U,STATE0,STATE1);
  signal state: StateType;

begin
  process(clk50,reset)
  begin
    if reset='1' then
      state <= STATE0;
    elsif rising_edge(clk50) then
      case state is
        when STATE0 =>
          y <= x;
          state <= STATE1;
        when others =>    -- STATE1
          y <= not x;
          state <= STATE0;
      end case;
    end if;
  end process;

end architecture;
```

Nytt: `type` definierar en egen datatyp



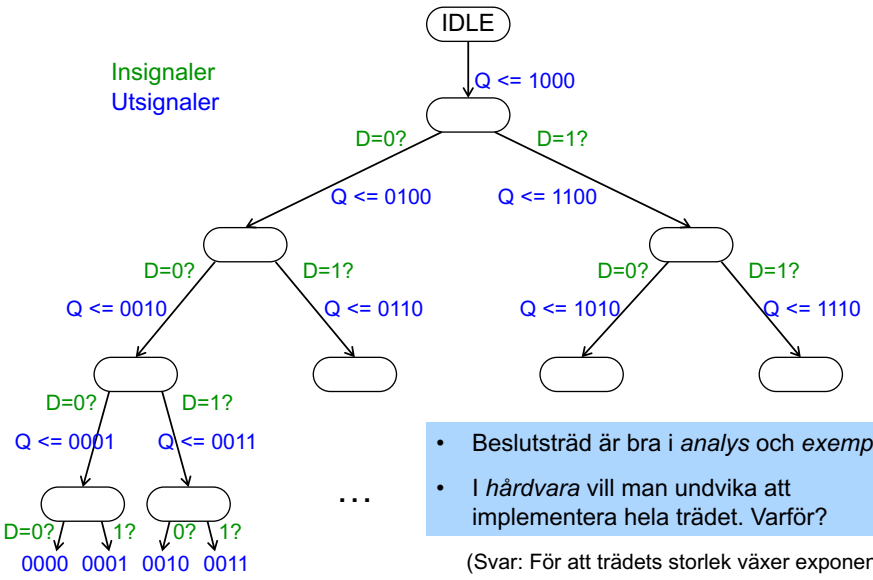
3. Digital implementation av SAR



1. Gissa på $V_{max}/2$ 10000…
2. För lite ⇒ öka med $V_{max}/4$ 11000…
   För mycket ⇒ minska med $V_{max}/4$ 01000…
3. För lite ⇒ öka med $V_{max}/8$
   För mycket ⇒ minska med $V_{max}/8$

…

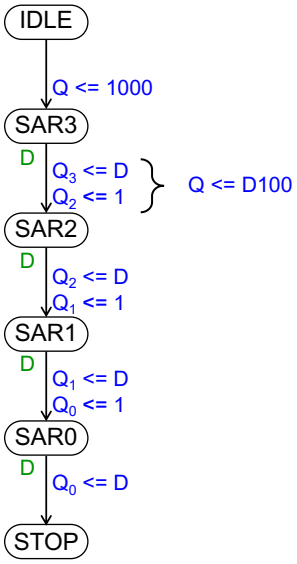
Beslutsträd



- Beslutsträd är bra i *analys* och *exempel*
- I *hårdvara* vill man undvika att implementera hela trädet. Varför?

(Svar: För att trädets storlek växer exponentiellt)

Tillståndsdiagram – bättre

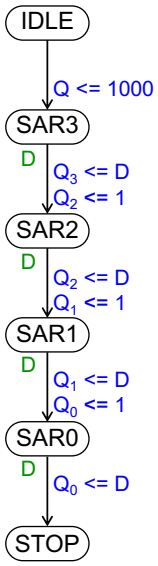


VHDL-kod

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

entity ADC is port(
  clk50, reset: in std_logic;
  D: in std_logic;    -- input from comparator: higher or lower?
  Q: out std_logic_vector(3 downto 0));    -- parallel output
end entity;

architecture state_machine of ADC is
  type StateType is (U,IDLE,SAR3,SAR2,SAR1,SAR0,STOP); -- list of states
  signal state: StateType;

begin
  process(clk50,reset)
  begin
    ...
  end process;

end architecture;
```

se nästa sida

```
begin
  process(clk50,reset)
  begin
    if reset='1' then
      state <= IDLE;
    elsif rising_edge(clk50) then
      case state is
        when IDLE =>
          Q <= "1000";   -- half of the maximum value
          state <= SAR3;
        when SAR3 =>
          Q(3) <= D;   -- MSB
          Q(2) <= '1';
          state <= SAR2;
        when SAR2 =>
          Q(2) <= D;
          Q(1) <= '1';
          state <= SAR1;
        when SAR1 =>
          Q(1) <= D;
          Q(0) <= '1';
          state <= SAR0;
        when SAR0 =>
          Q(0) <= D;
          state <= U;
        when others =>  -- STOP state
          -- nothing
      end case;
    end if;
  end process;
```



Läs & lös

Läs:

- Bengtsson s 154–162 och 165–168

  (U_{ref} skall vara $-U_{ref}$ i ekvationerna på s 158–159, eftersom U_{ref} <0)

Lös:

- Övningsuppgift SAR (nästa sida)
- Bengtsson uppg 4.10, 4.11, 4.13 och 4.16

Övningsuppgift SAR

En 4-bitars A/D-omvandlare accepterar inspänningar i intervallet 0–7.5 V utan att det blir overflow. Den fungerar enligt SAR-principen.

(a) Vad blir den digitala utsignalen om insignalen är 5.8 V?

(b) Vilken sekvens av digitala gissningar Q kommer SAR-kretsen att generera för en insignal på 5.8 V, och vilka utsignaler D kommer komparatorn att ge för vart och ett av dessa Q-värden? Komplettera beslutsträdet på bild 26 och rita in den aktuella vägen.

(c) Följ tillståndsdiagrammet på bild 27 för samma insignal. Ange Q-värdet i varje steg och verifiera att utsignalen till slut blir samma som i (a) och (b).

4.10 En dual slope-AD liknande den i figur 4.63, har en binärräknare bestående av 10 bitar och en referensspänning på −5 V. Klockan till binärräknaren går med 1 MHz.

a) Inom vilket spänningsområde får A_{in} ligga? (Här är $A_{in} = V_{in}$)

b) Hur lång tid skulle AD-omvandlingen ta om $A_{in} = 3$ V?

c) Rita upp- och urladdningskurvan om $RC = 1$ ms (och $A_{in} = 3$ V).

d) Anta att resistorns och kondensatorns värden ändras (på grund av temperaturvariationer eller åldrande) så att $RC = 0{,}9$ ms. Rita upp- och urladdningskurvan även i detta fall. Hur påverkas resultatet av AD-omvandlingen av detta?

4.11 Spänningen 2,84 V ska AD-omvandlas i en 6-bitars successiv approximationsomvandlare med referensspänningen 5 V (vilket innebär att DA-omvandlaren har en referensspänning på –5 V).

a) Hur många jämförelser görs innan AD-omvandlingen är klar?

b) Rita ett tidsdiagram som visar DA-omvandlarens utsignal under omvandlingen.

c) Vad blir resultatet av omvandlingen och hur stort blir felet?

4.13 Figur 4.65 visar de möjliga spänningsnivåerna vid de olika jämförelserna i en successiv approximations-AD.

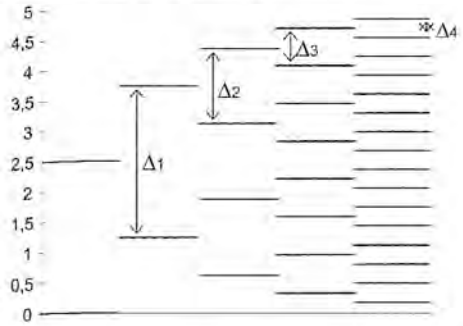


Fig 4.65 Alla möjliga jämförelsenivåer

a) Hur många bitar består AD-omvandlaren av?

b) Vilken referensspänning har den?

c) Beräkna Δ_1 - Δ_9.

d) Hur stor är AD-omvandlarens upplösning?

e) Vilken utsignal skulle AD-omvandlaren ge för insignalen 2,1 V? Gör inga beräkningar utan använd en linjal och mät i figur 4.65

4.16a) Hur många komparatorer består AD-omvandlaren i figur 4.67 av?

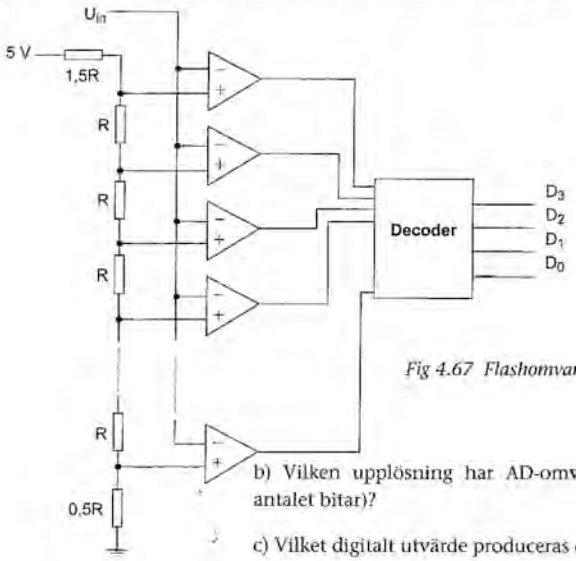


Fig 4.67 Flashomvandlare

b) Vilken upplösning har AD-omvandlaren ovan (uttryckt i volt och antalet bitar)?

c) Vilket digitalt utvärde produceras om $U_{in} = 3{,}79$ volt?

d) Inom vilket intervall ligger U_{in} om $D_{ut} = 12$?

Föreläsning 8
Tillståndsmaskiner och seriell transmission

Erik Agrell 2017-09-13

Innehåll:

1. SAR-omvandlaren, forts.
2. Sample-and-hold
3. Seriell transmission
4. Mer om tillståndsmaskiner
5. Synkronisering av asynkron insignal

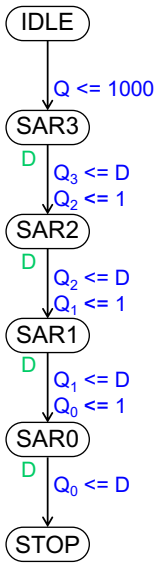
Bilder av E. Agrell och A. Linde

1. SAR-omvandlaren, forts.

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

entity ADC is port(
  clk50, reset: in std_logic;
  D: in std_logic;    -- input from comparator: higher or lower?
  Q: out std_logic_vector(3 downto 0));    -- parallel output
end entity;

architecture state_machine of ADC is
  type StateType is (U,IDLE,SAR3,SAR2,SAR1,SAR0,STOP); -- list of states
  signal state: StateType;

begin
  process(clk50,reset)
  begin
    ...
  end process;

end architecture;
```

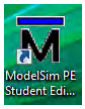
se nästa sida

```
begin
  process(clk50,reset)
  begin
    if reset='1' then
      state <= IDLE;
    elsif rising_edge(clk50) then
      case state is
        when IDLE =>
          Q <= "1000";   -- half of the maximum value
          state <= SAR3;
        when SAR3 =>
          Q(3) <= D;   -- MSB
          Q(2) <= '1';
          state <= SAR2;
        when SAR2 =>
          Q(2) <= D;
          Q(1) <= '1';
          state <= SAR1;
        when SAR1 =>
          Q(1) <= D;
          Q(0) <= '1';
          state <= SAR0;
        when SAR0 =>
          Q(0) <= D;
          state <= U;
        when others =>  -- STOP state
          -- nothing
      end case;
    end if;
  end process;
```



Simulering av SARen

Demo



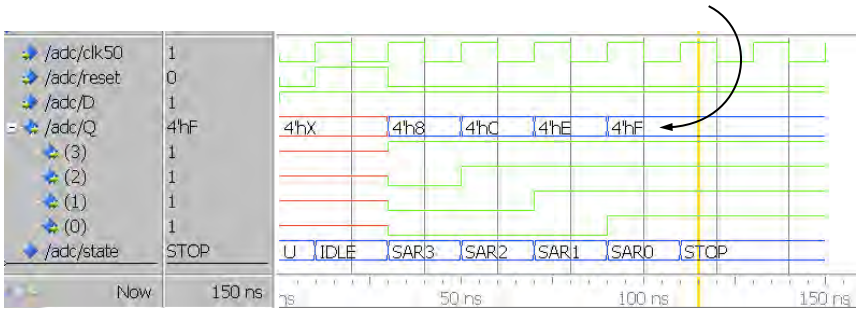
Vi kan inte simulera hela A/D-omvandlaren i ModelSim, eftersom den innehåller både digital och analog elektronik.

Antag att V_{in} är lika med maxspänningen. Då ger komparatorn D = 1 varje gång.

Exempel på do-fil:

```
# standard header:
vsim work.ADC
view wave
add wave *
#restart -force

# simulation commands:
force clk50 0 0ns, 1 10ns -repeat 20ns
force reset 0 0ns, 1 10ns, 0 30ns
# assume analog input is maximum:
force D 1

run 150ns
```

Gissningarna är i tur och ordning 8, 12, 14,15



Efter A/D-omvandling är Q = 1111

Tips för praktiskt användbar A/D

① Återvänd till IDLE efter STOP
- annars blir det bara en enda konvertering

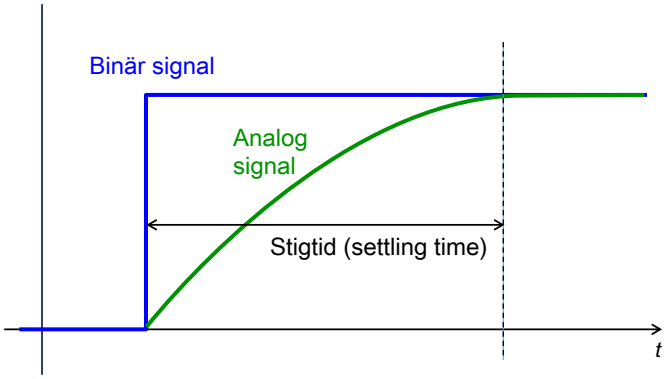
② SARen behöver två periodiska triggsignaler
- en långsam som sätter igång A/D-konverteringen av ett sampel och en snabbare som skiftar tillståndsmaskinen ett steg

③ Trigga SARen långsammare än i exemplet
- annars hinner inte D/A-omvandlaren konvergera, se nästa bild

④ Använd fler bitar än i exemplet
- annars blir signalkvaliteten dålig

Har detta med lab 3 att göra?

D/A-omvandlarens snabbhet



Typiskt stigtid är ca 150 ns

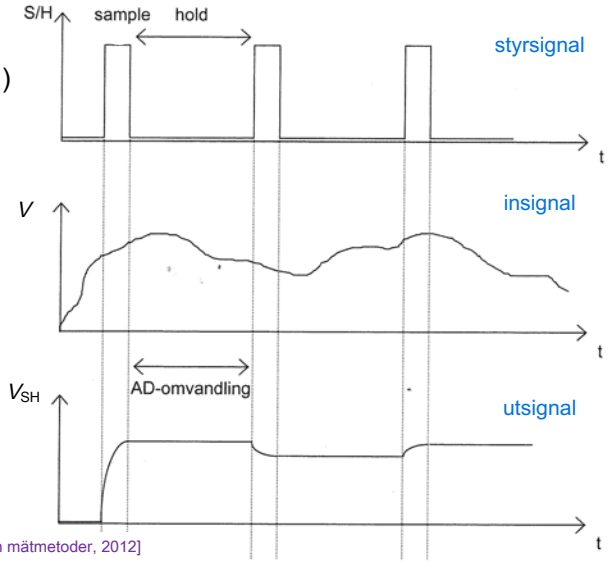
Vad innebär detta för SAR-omvandlaren?

A/D-omvandlarens snabbhet

- D/A-omvandlaren behöver ca 150 ns
- En *n*-bitars A/D-omvandlare enligt SAR-principen behöver därför ca $n \cdot 150$ ns
- Detta ger en övre gräns för frekvensen hos SAR-kretsens triggsignaler
- En 50 MHz-oscillator (20 ns mellan pulserna) är alldeles för snabb

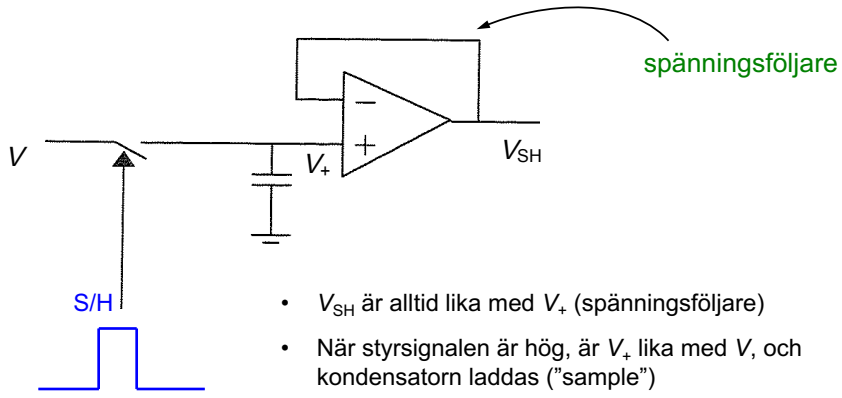
2. Sample-and-hold

En sample-and-hold (S/H) krets ”fryser” en analog spänning en stund.



[Bengtsson: Elektriska mätsystem och mätmetoder, 2012]

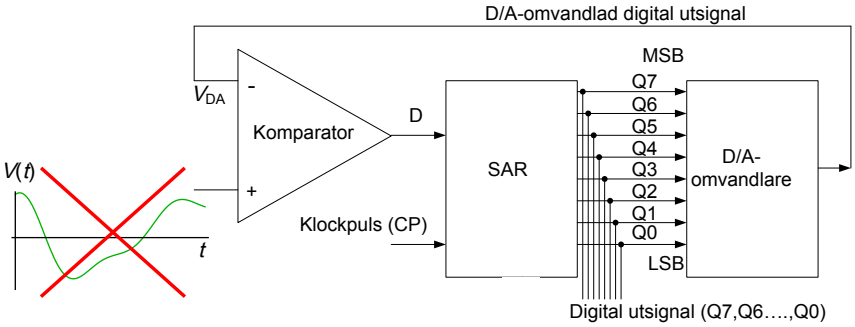
Principen för sample-and-hold



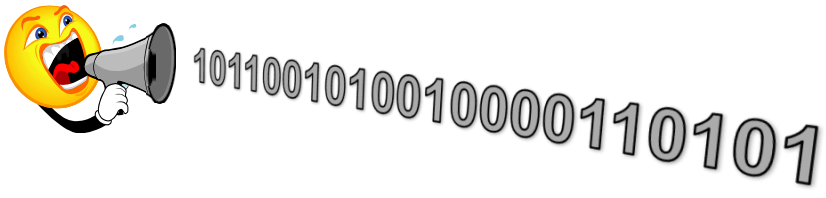
- V_{SH} är alltid lika med V_+ (spänningsföljare)
- När styrsignalen är hög, är V_+ lika med V, och kondensatorn laddas (”sample”)
- När styrsignalen är låg, behåller kondensatorn spänningen V_+ (”hold”)

När har man nytta av en S/H-krets?

Svar: Exempelvis vid A/D-omvandling.
A/Dn behöver en stabil insignal.

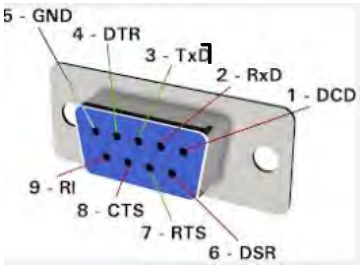


3. Seriell transmission



RS-232-standarden

- Standardiserat protokoll för seriell datakommunikation
- Definierades 1962
- Föregångaren till USB

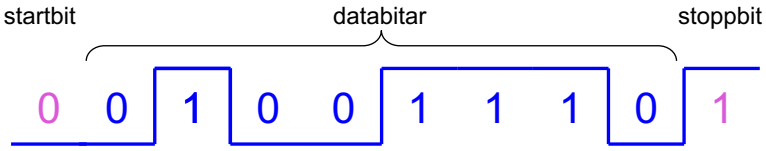


- 9-polig standardkontakt
- Logisk 0 motsvarar hög spänning och 1 är låg
- För varje byte (8 databitar) skickas 1 start- och 1 stoppbit
- LSB skickas först

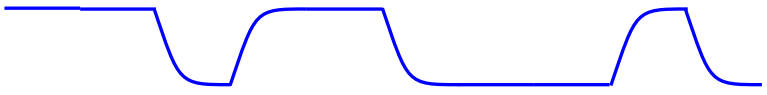
- Flexibla nivåer: från 3 till 15 V för 0 och från –15 till –3 V för 1
- Flexibel datahastighet

Logiska och elektriska nivåer

- Logisk nivå (0 eller 1), syns i ModelSim:

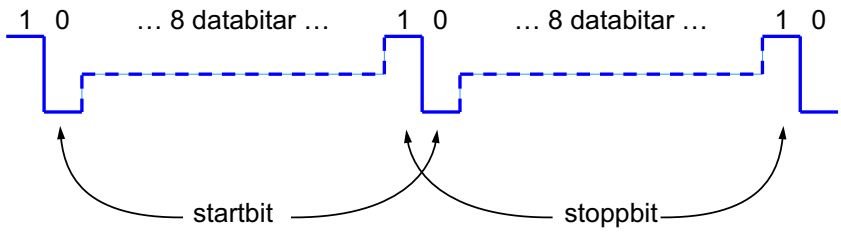


- Elektrisk nivå (spänning), syns på oscilloskop:



Blanda inte ihop dem!

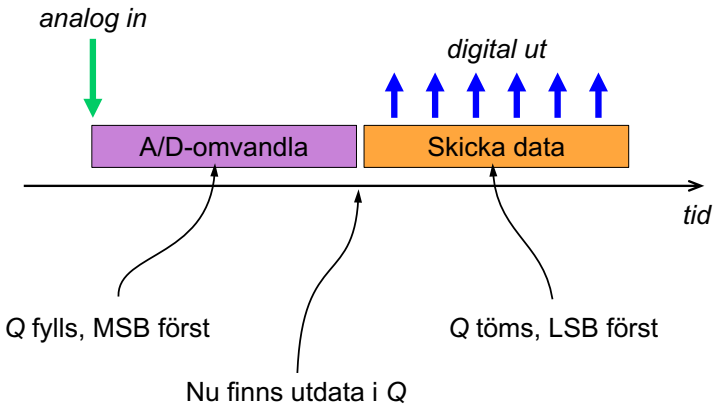
Start- och stoppbitar enligt RS-232



Alltid positiv flank mellan stopp- och startbit

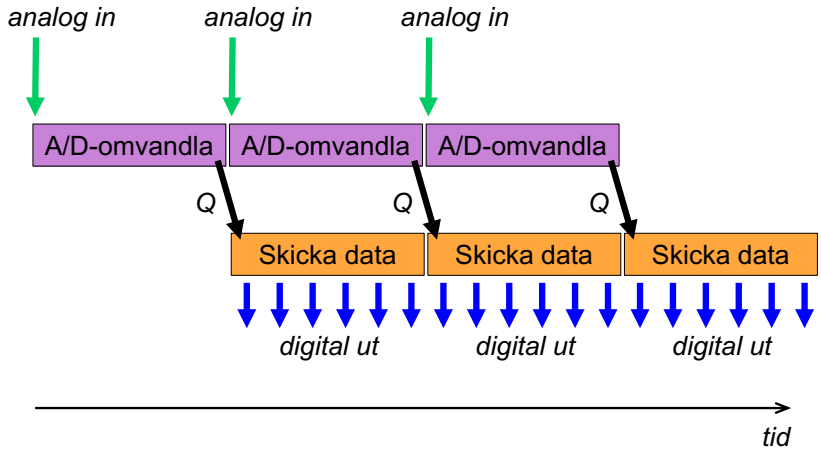
⇒ underlättar synkronisering

Kombination av A/D och seriell transmission



Lab 3 Lab 1

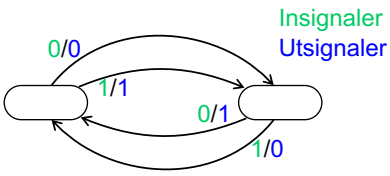
Upprepad A/D-omvandling



4. Mer om tillståndsmaskiner

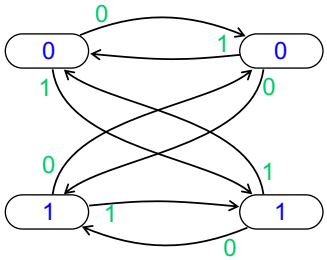
Det finns två typer av tillståndsmaskiner. Exempel (från F7 bild 23):

Mealy



in: 00001111
ut: 01011010

Moore

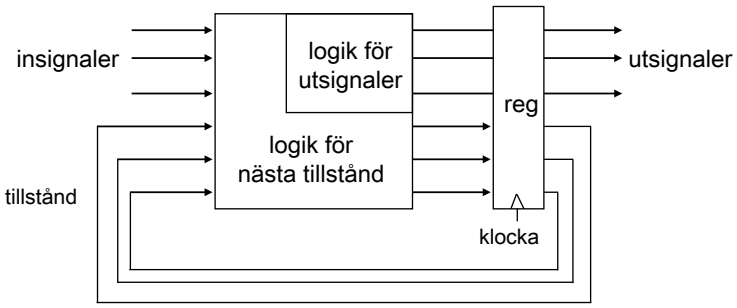


in: 00001111
ut: 01011010

De båda typerna realiserar samma funktion (invertera varannan bit)

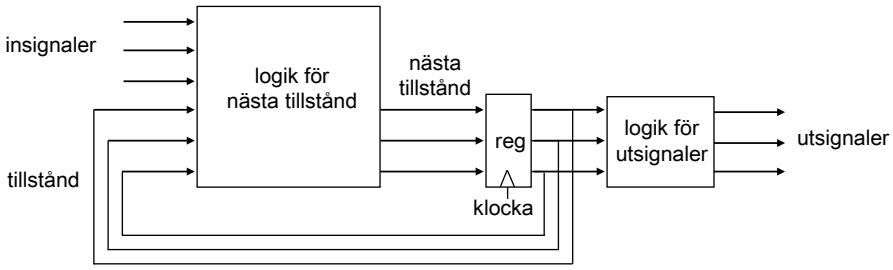
Principschema, Mealy

Mealy: utsignalerna beror på tillståndet och insignalerna



Principschema, Moore

Moore: utsignalerna beror endast på tillståndet

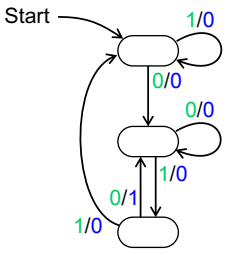


- Moore-typen kräver ofta fler tillstånd än Mealy.
- Mealy-typen finns i två varianter, synkron och asynkron. I den här kursen använder vi synkron Mealy.

Mönsterigenkänning

- Mönsterigenkänning innebär att leta efter ett visst mönster i en insignal
- *Problem:* En synkron krets skall konstrueras för att söka efter sekvensen ”010” i en insignal. Varje gång detta inträffar skall utsignalen vara 1, annars noll.

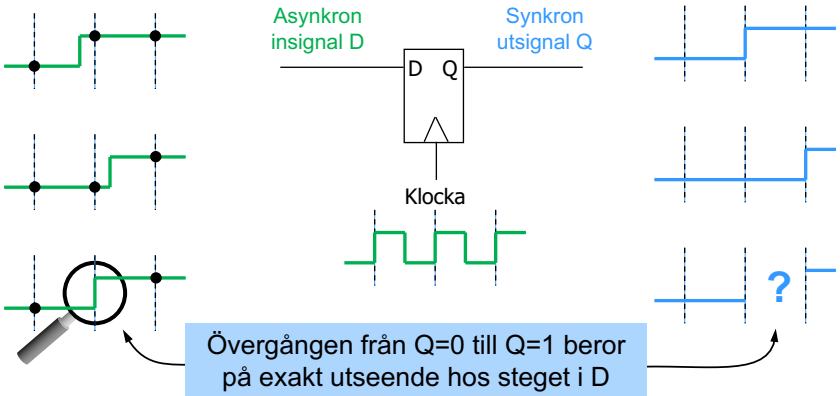
In: 00010011010101
Ut: 00001000001010



□

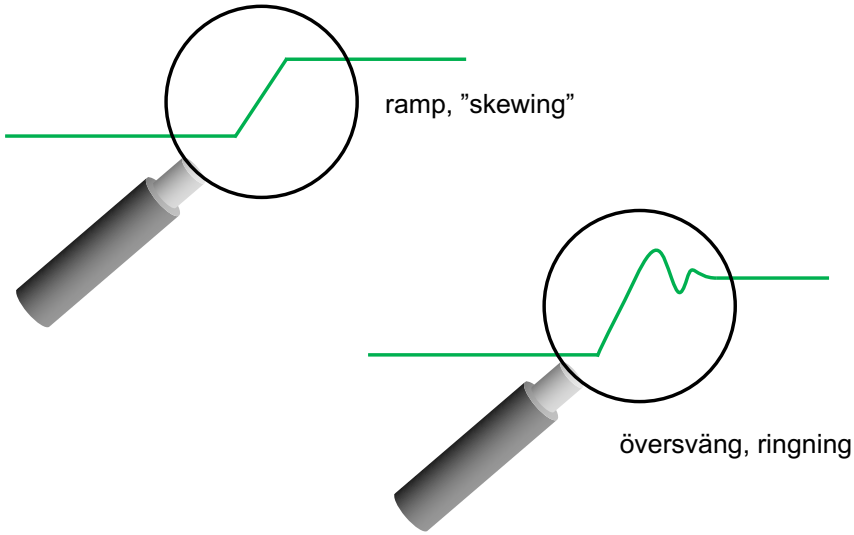
5. Synkronisering av asynkron insignal

I mötet mellan asynkront och synkront kan konstiga saker hända…



Övergången från Q=0 till Q=1 beror på exakt utseende hos steget i D

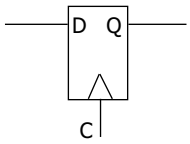
Språng i verkligheten

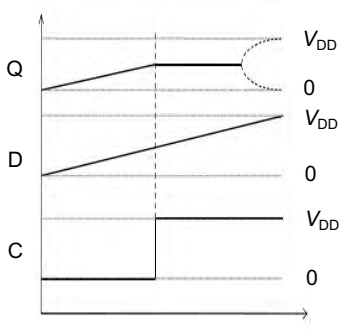


Risk 1: metastabilitet

Digitala signaler kan i hårdvara inte hoppa momentant mellan '0' och '1'.

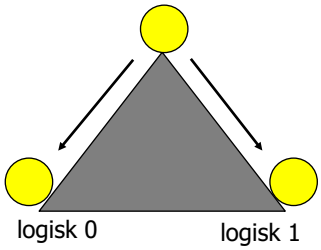
- Antag att insignalen D till vippan slår om långsamt i förhållande till klockan C, och att C slår om precis när D är vid $V_{DD}/2$.
- Då låser sig vippan vid ett mellanläge. Efter en tid slår vippan om till antingen '0' eller '1'.
- Fenomenet kallas metastabilitet

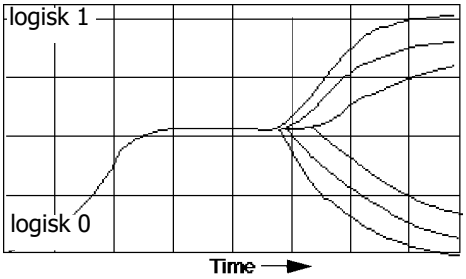




Metastabilitet uppstår när vippans ingång ändras nära klockans flank

- vippan hamnar i ett nästan stabilt (d.v.s ”metastabilt”) tillstånd – varken 0 eller 1
- den kan stanna i detta tillstånd obegränsat länge
- osannolikt men inte omöjligt

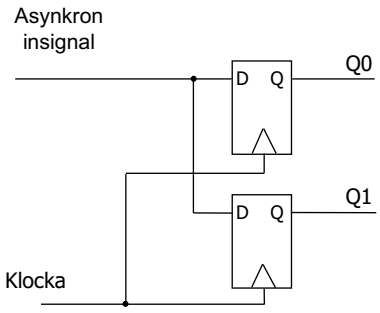




Hur det kan se ut på ett oscilloskop när metastabilitet uppträder

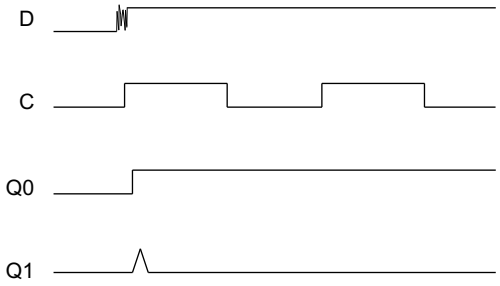
Asynkron insignal till två vippor

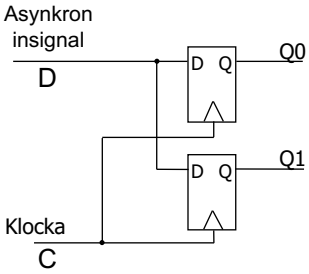
Ännu värre kan det bli om samma asynkrona insignal går till två vippor



Här förväntar man sig att Q0=Q1 alltid, eller hur?

Risk 2: Otillåtet tillstånd





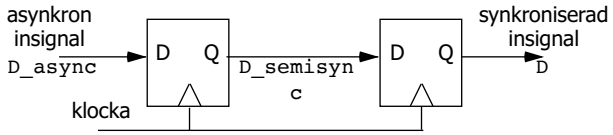
En vippa tolkar D som 0, den andra som 1

⇒ Q0 ≠ Q1, otillåtet tillstånd!

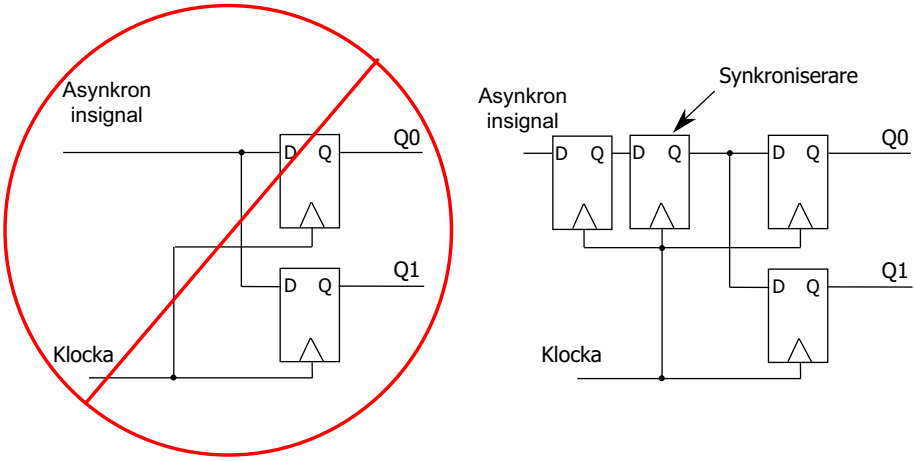
Synkronisering av asynkron ingång

- Risken för metastabilitet och förbjudna tillstånd kan aldrig elimineras helt, men den kan minskas radikalt
  - Använd långsammare systemklocka, vilket ger vippan mer tid för att nå ett stabilt tillstånd
  - Använd snabbast möjliga logikteknologi
  - Klocka den asynkrona signalen, gärna två gånger

```
process(clk50)
  begin
    if rising_edge(clk50) then
      D_semisync <= D_async;
      D <= D_semisync;
    end if;
  end process;
```



- Aldrig låta asynkrona insignaler att vara kopplade till mer än en vippa
  - synkronisera så snart som möjligt och behandla därefter signalen som synkron



Läs & lös

Nästa föreläsning: 22 sept i J121

Läs:

- Bengtsson avsnitt 4.4 (sample-and-hold)
- ”Kretskonstruktion med VHDL”, avsnitt 8
- Lab-PM avsnitt 3

Lös:

- Använd en slumpmässig sekvens med 10 bitar som insignal till tillståndsmaskinerna på bild 25 och verifiera att de ger samma utsignal
- Förberedelseuppgifter till Lab 3 (jippi, SAR!)
- Inlämningsuppgift 2
- Förbeberedelse till Lab 4: uppg 4.1–4.3, 4.6

Föreläsning 9
Synkronisering, mottagare

Erik Agrell 2017-09-22

Innehåll:

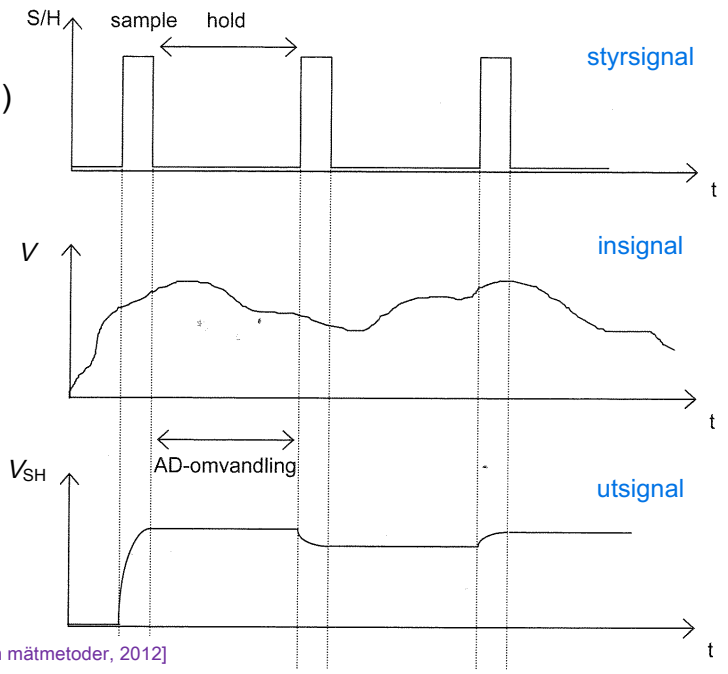
1. Mer om sample-and-hold
2. Synkronisering i mottagare
3. Seriell mottagare och sampling

Bilder av E. Agrell

1. Mer om sample-and-hold

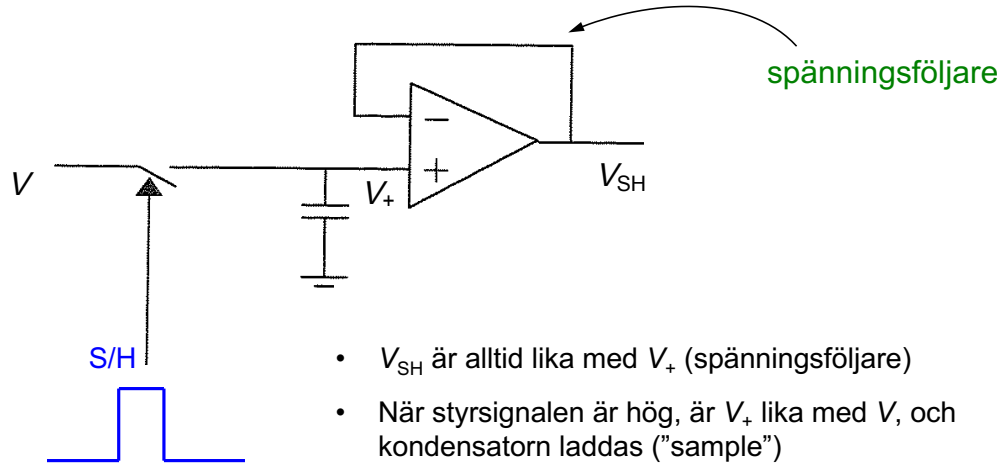
Från F8:

En sample-and-hold (S/H) krets ”fryser” en analog spänning en stund.



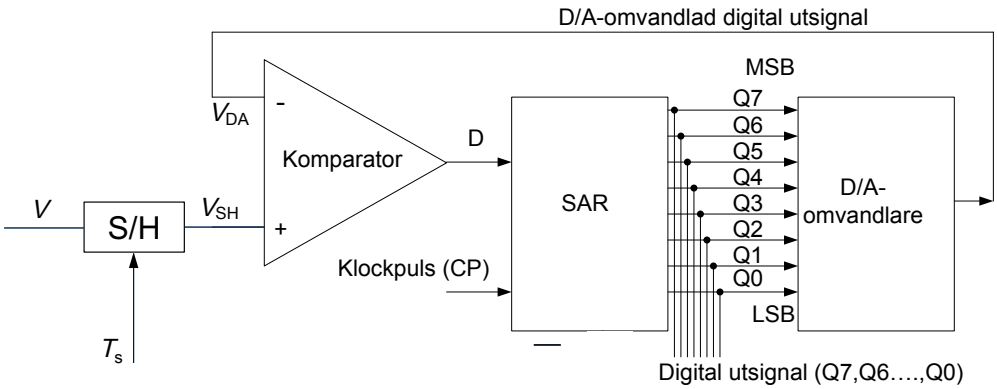
[Bengtsson: Elektriska mätsystem och mätmetoder, 2012]

Principen för sample-and-hold

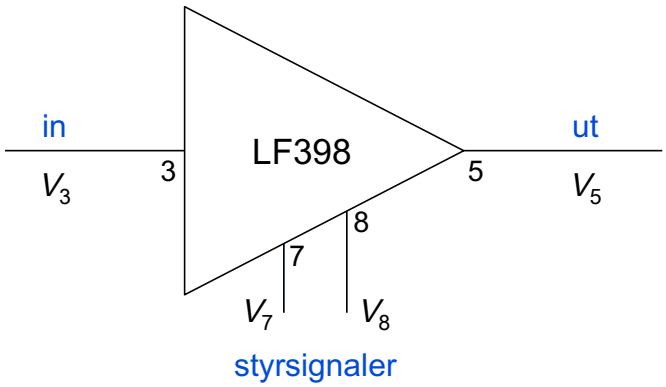


- V_{SH} är alltid lika med V_+ (spänningsföljare)
- När styrsignalen är hög, är V_+ lika med V, och kondensatorn laddas (”sample”)
- När styrsignalen är låg, behåller kondensatorn spänningen V_+ (”hold”)

Inkoppling till A/D-omvandlaren



Styrning av S/H-kretsen LF398

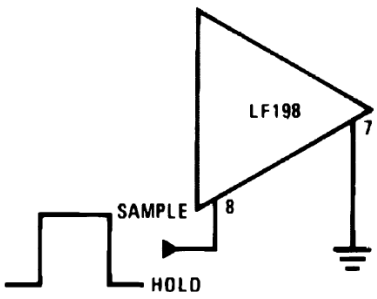


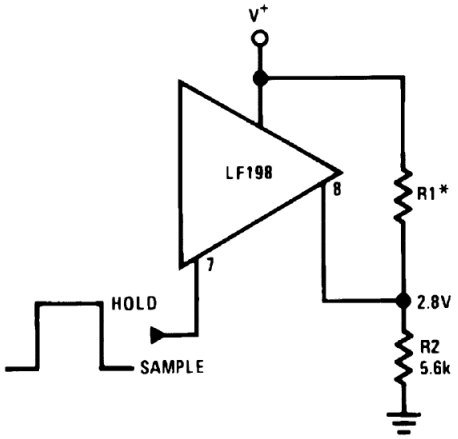
- Sample om $V_8 - V_7 > 1.4$ V $\Rightarrow V_5 = V_3$
- Hold om $V_8 - V_7 < 1.4$ V $\Rightarrow V_5$ behåller tidigare V_3

Dimensionering av styrsignalen

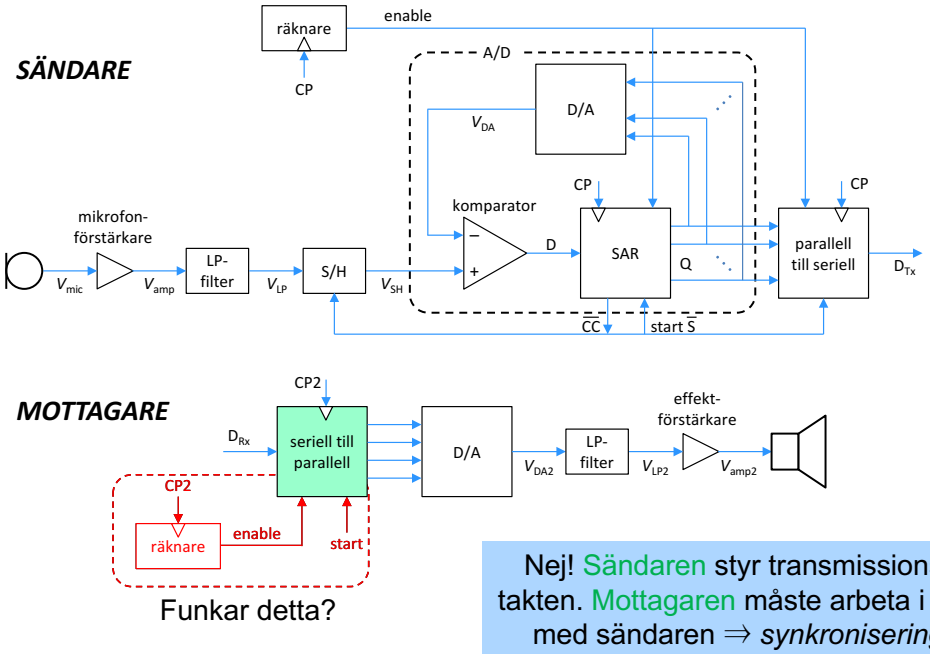
- Vill man sampla vid positiv puls, läggs styrsignalen på pinne 8
- Vill man sampla vid negativ puls, läggs styrsignalen på pinne 7
- Den andra pinnen ges en konstant referensspänning

Exempel ur datablad:





2. Synkronisering i mottagare

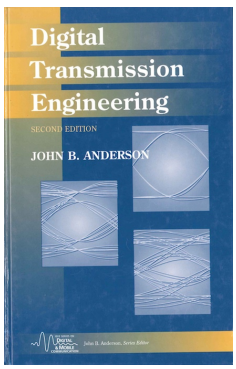


Funkar detta?

Nej! Sändaren styr transmissionstakten. Mottagaren måste arbeta i takt med sändaren ⇒ *synkronisering*

important and challenging medium, the fading mobile channel. Some further material appears in Chapter 7.

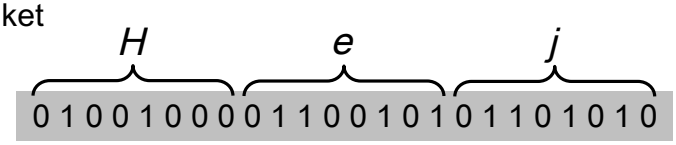
Probably the most underrated area in transmission design is synchronization. Little time is spent in most books on this subject, yet synchronization of various types probably consumes half of transmission design effort. In reality, there are several layers to the synchronization of a system: An RF system needs receiver carrier synchronization, all systems need to identify the symbol boundaries, and most data



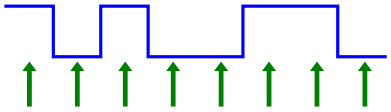
[John B. Anderson, *Digital Transmission Engineering*]

Typer av synkronisering

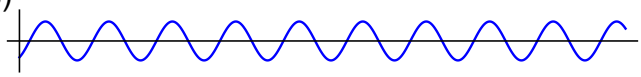
- Blocksynkronisering är att gruppera bitar till bytes och bytes till datapaket



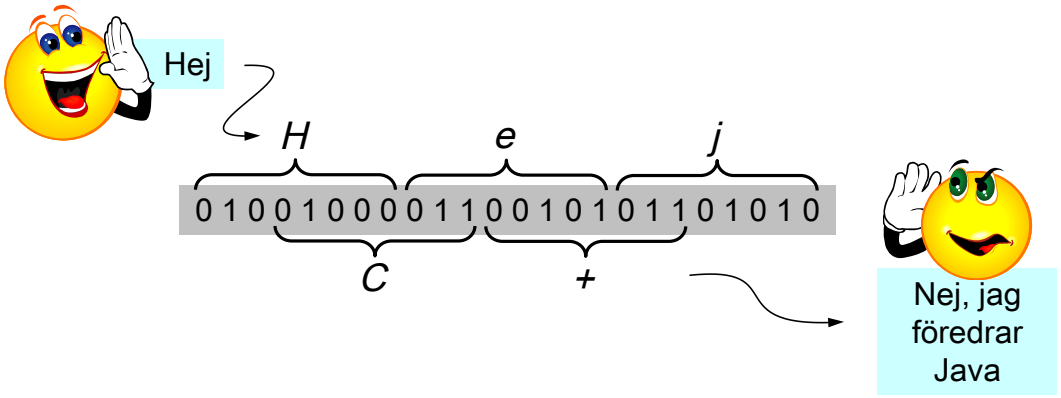
- Symbolsynkronisering är att välja samplingsögonblick i varje symbol (bit)



- Fassynkronisering är att identifiera fasläget hos en sinusvåg, vilket behövs om fasen bär information (fasmodulation, inte i denna kurs)

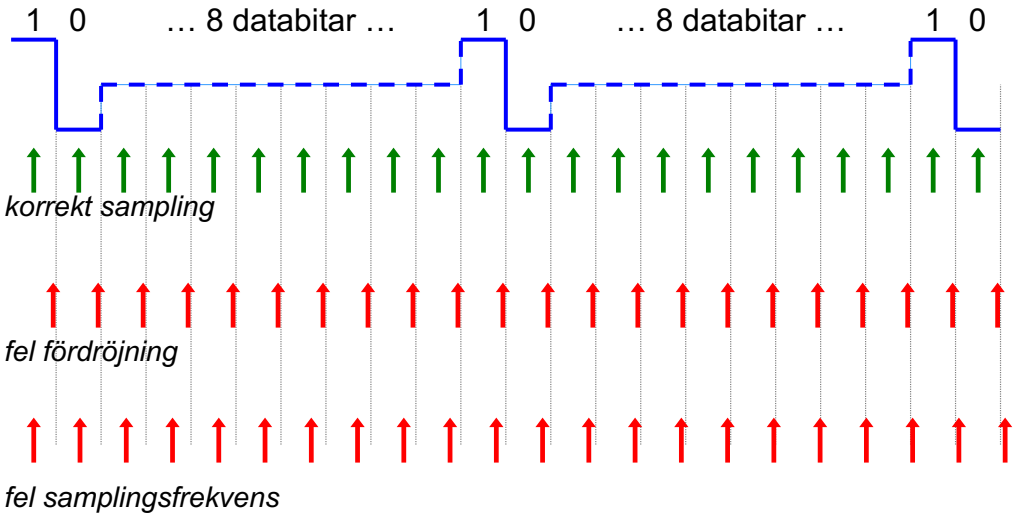


Felaktig blocksynkronisering



Åtgärd: Använd ett protokoll med redundans, t.ex. start- och stoppbitar eller pakethuvud (”preamble”)

Felaktig sampling (symbolsynkronisering)



Att motverka samplingsfel

Hur kan sändare och mottagare hålla samma bithastighet?

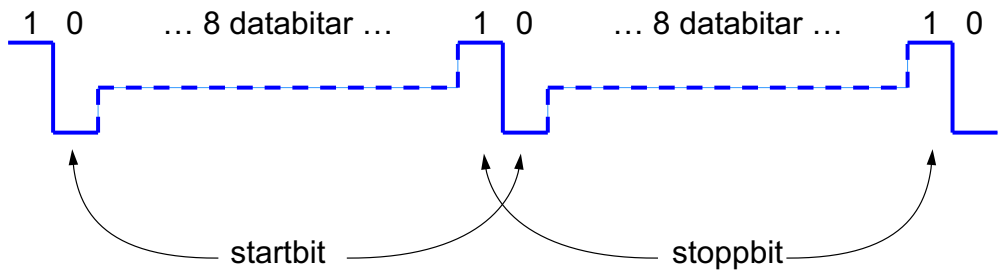
1. *Noggranna oscillatorer?* Nej, även bra kristalloscillatorer driver något.
2. *Använda sändarklockan även i mottagaren?* OK, men det kräver en sidokanal.
3. *Återvinna sändarklockan ur mottagen signal?* OK, om signalkvaliteten är hyfsad. ⇒ symbolsynkronisering

I kommunikationssystem (och i labben) är metod 3 vanligast

3. Seriell mottagare och sampling

Tidssignal enligt RS-232 (från F8):

(Här visas 1 som hög nivå och 0 som låg nivå, trots att 1 enligt RS-232-standarden är *låg* spänning.)



Alltid samma flank 1→0 mellan stopp- och startbit

⇒ underlättar synkronisering

Symbolsynkronisering

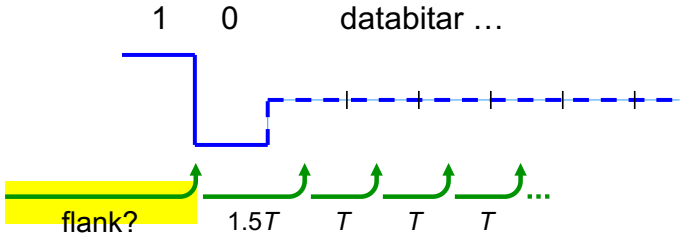
Klockåtervinning = ”clock recovery” = ”symbol synchronization”:

Att hitta rätt samplingstidpunkter för den mottagna signalen

Grundprincip:

1. Sök flank
2. Vänta 1.5T, sampla
3. Vänta T, sampla
4. Vänta T, sampla
5. …

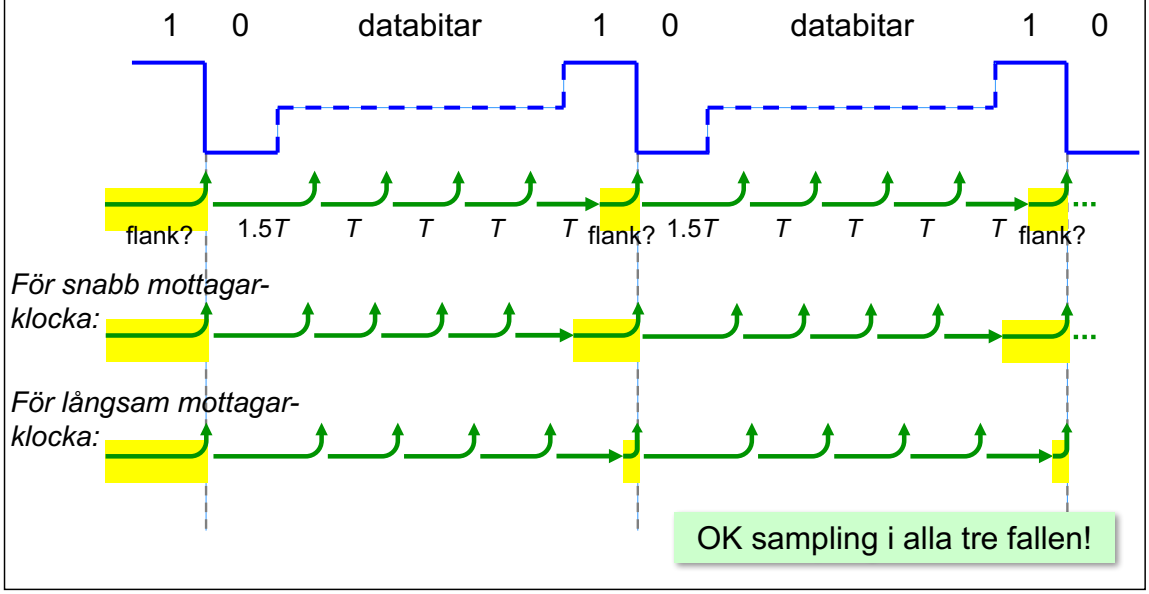
T = symboltiden = 1/baudrate



Klockdrift

Sändar- och mottagaroscillatorerna är inte helt synkrona

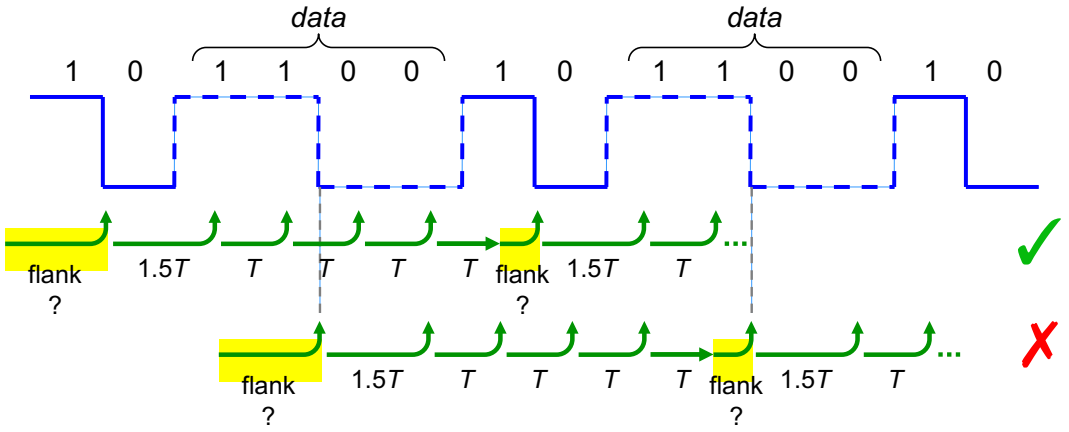
⇒ trigga på varje startbit



OK sampling i alla tre fallen!

Problem: flanker i data

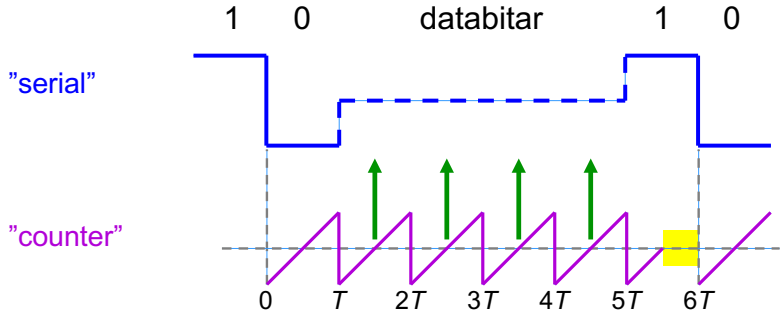
Regelbundna flanker i dataorden kan få mottagaren att trigga fel:



- Mottagaren kan låsa på två sätt – inte bra!
- Detta händer bara om data upprepar sig
- Vanligt fenomen vid test och uppstart av system
- Ovanligt vid verklig transmission (varför?)

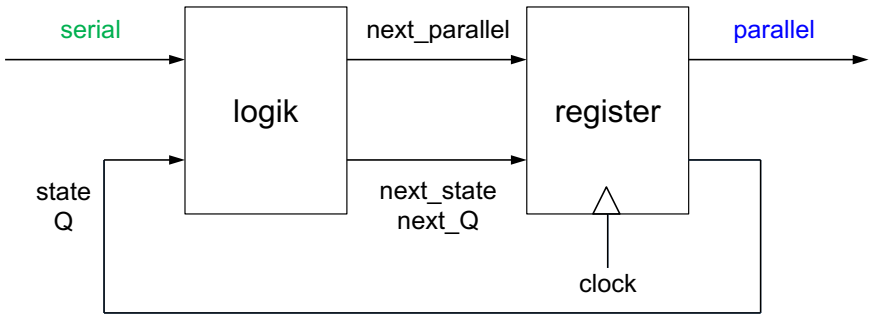
Implementering av mottagaren

För att implementera en fördröjning behövs en klocka och en räknare

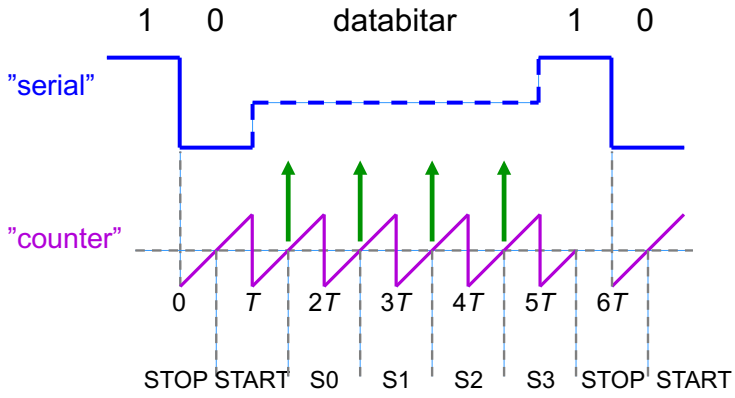


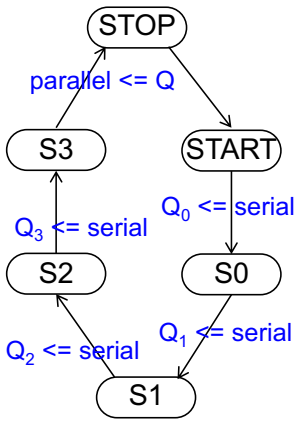
- Nollställ räknaren vid varje flank 1→0
- Nollställ räknaren när den når det värde som motsvarar *T*
- Sampla när räknaren är vid *halva* maxvärdet
- Börja vänta på nästa flank *T* tidsenheter efter sista datasamplingen
- Räkna bitar med hjälp av en tillståndsmaskin

Tillståndsmaskin enligt Mealy



Vi behöver ett tillstånd för varje bit

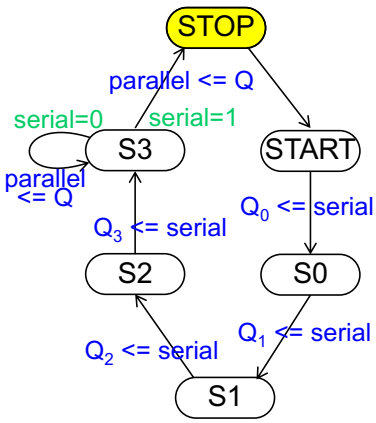




- Nytt tillstånd varje gång räknaren är vid halva maxvärdet
- Tillståndsmaskinen gör också serie-till-parallell-omvandling

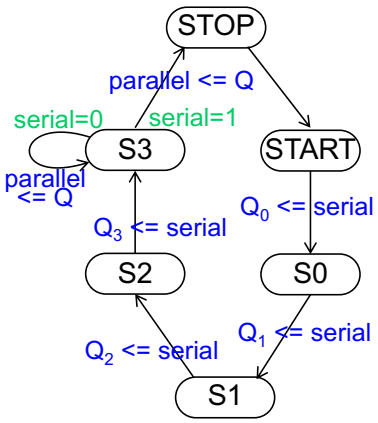
Tillståndsmaskin med synkronisering

- Tillståndens längd styrs av klockan, som är inställd på *ungefär* rätt bithastighet
- Efter S3 kommer stoppbit (serial=1). Annars väntar man en bit ⇒ blocksynkronisering
- Mitt i STOP-tillståndet väntar man in flanken 1→0 ⇒ symbolsynkronisering

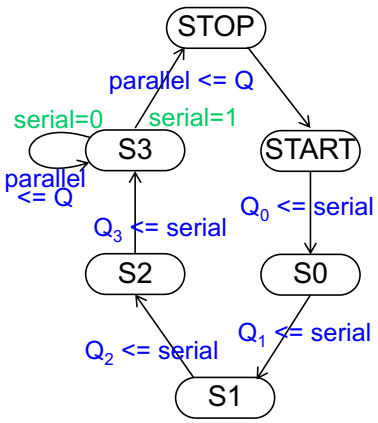


Tillståndsmaskin i VHDL

```
process(state,serial,Q)   -- update state and outputs
begin
  next_Q <= Q;
  case state is
    when STOP =>
      next_state <= START;
    when START =>
      next_state <= S0;
      next_Q(0) <= serial;
    when S0 =>
      next_state <= S1;
      next_Q(1) <= serial;
    when S1 =>
      next_state <= S2;
      next_Q(2) <= serial;
```



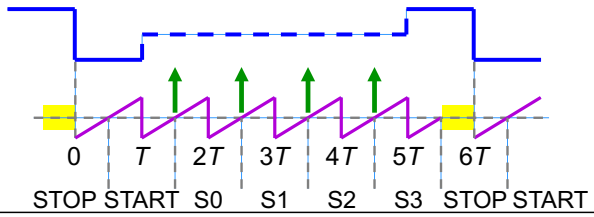

```
    when S2 =>
      next_state <= S3;
      next_Q(3) <= serial;
    when S3 =>
      next_parallel <= Q;
      if serial='0' then
        next_state <= S3;
      else
        next_state <= STOP;
      end if;
    when others =>   -- undefined state
      next_state <= STOP;
  end case;
end process;   -- update state

end architecture;
```



```
process(clk50,reset)    -- bit clock, sync'd to start bit
begin
  if reset='0' then
    state <= STOP;
    Q <= (others => '0');
    counter <= (others => '0');
  elsif rising_edge(clk50) then
    counter <= counter+1;
    if state=STOP and serial='1' then
      counter <= (others => '0');
    elsif counter=4 then    -- half-period 5 clock cycles
      state <= next_state;
      Q <= next_Q;
      parallel <= next_parallel;
    elsif counter=9 then    -- bit clock 50MHz/(9+1) = 5 MHz
      counter <= (others => '0');
    end if;
  end if;
end process;    -- bit clock
```

Antag att klockan är 50 MHz och önskad bithastighet 5 Mbit/s

T = 10 klockcykler för varje bit



VHDL-kod för mottagaren

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

entity Rx is port(
  reset, clk50: in std_logic;
  serial: in std_logic;
  parallel: out std_logic_vector(3 downto 0));
end entity;

architecture state_machine of Rx is
  type StateType is (U,STOP,START,S0,S1,S2,S3);
  signal state, next_state: StateType;
  signal Q, next_Q, next_parallel: std_logic_vector(3
downto 0);
  signal counter: std_logic_vector(3 downto 0);
begin

process(clk50,reset)   -- bit clock, sync'd to start
bit
begin
  if reset='0' then
    state <= STOP;
    Q <= (others => '0');
    counter <= (others => '0');
  elsif rising_edge(clk50) then
    counter <= counter+1;
    if state=STOP and serial='1' then
      counter <= (others => '0');
    elsif counter=4 then   -- half-period 5 clock
cycles
      state <= next_state;
      Q <= next_Q;
      parallel <= next_parallel;
    elsif counter=9 then   -- bit clock 50MHz/(9+1) = 5
MHz
      counter <= (others => '0');
    end if;
  end if;
end process;   -- bit clock

process(state,serial,Q)   -- update state and outputs
begin
  next_Q <= Q;
  case state is
    when STOP =>
      next_state <= START;
    when START =>
      next_state <= S0;
      next_Q(0) <= serial;
    when S0 =>
      next_state <= S1;
      next_Q(1) <= serial;
    when S1 =>
      next_state <= S2;
      next_Q(2) <= serial;
    when S2 =>
      next_state <= S3;
      next_Q(3) <= serial;
    when S3 =>
      next_parallel <= Q;
      if serial='0' then
        next_state <= S3;
      else
        next_state <= STOP;
      end if;
    when others =>   -- undefined state
      next_state <= STOP;
  end case;
end process;   -- update state

end architecture;
```

Exempel på DO-fil

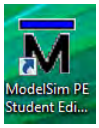
```
# header:
vsim work.rx; view wave
add wave clk50 reset serial state counter parallel

# clock and reset signals:
restart -force
force clk50 0 0ns, 1 10ns -repeat 20ns
force reset 0 0ns, 1 50ns

# serial is a repeated sequence of:
#   0 (start bit)
#   1100 (data bits)
#   1 (stop bit)
force serial 0 0,   1 200, 1 400, 0 600, 0 800,
1 1000 -repeat 1200

run 2400ns
wave zoom full
```

Tid för en bit:
T = 10 klockcykler
= 200 ns

Data: 1100

Upprepa efter 6 bitar = 1200 ns

Demo



Regelbunden flank i data
⇒ synkroniseringsproblem

Mer realistisk simulering

```
# header:
#vsim work.rx; view wave
#add wave clk50 reset serial state counter parallel

# clock and reset signals:
restart -force
force clk50 0 0ns, 1 10ns -repeat 20ns
force reset 0 0ns, 1 200ns

# serial is a repeated sequence of:
#   0 (start bit)
#   1100 (data bits)
#   1 0 (stop and start bits)
#   1010 (next data bits)
#   1 (stop bit)
force serial 0 0,   1 200, 1 400, 0 600, 0 800,
1 1000, 0 1200,  1 1400, 0 1600, 1 1800, 0 2000,
1 2200 -repeat 2400

run 2400ns
wave zoom full
```

Första dataordet: 1100

Andra dataordet: 1010

12 bitar

Demo

Inför labben

VHDL-koden ovan är inte direkt användbar i hårdvara (lab 4), eftersom…

1. …bithastigheten är för hög (50 MHz/10 = 5 Mbit/s)…

```
signal counter: std_logic_vector(3 downto 0);
-- 4-bit counter
```

```
elsif counter=4 then    -- half-period 5 clock cycles
  state <= next_state;
  Q <= next_Q;
  parallel <= next_parallel;
elsif counter=9 then    -- bit clock 50MHz/(9+1) = 5 MHz
  counter <= (others => '0');
```


2. …ordlängden är för liten (4 bitar)…

```
parallel: out std_logic_vector(3 downto 0));
```

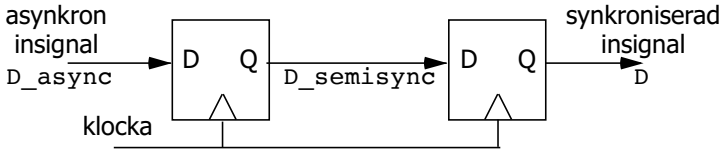
```
architecture state_machine of Rx is
  type StateType is (U,STOP,START,S0,S1,S2,S3);
  signal state, next_state: StateType;
  signal Q, next_Q, next_parallel: std_logic_vector(3 downto
0);
```

```
when S3 =>
  next_parallel <= Q;
  if serial='0' then
    next_state <= S3;
  else
    next_state <= STOP;
  end if;
```

3. …insignalen `serial` är asynkron och behöver göras synkron med hjälp av två vippor…

Exempel på en liknande process i sändaren (från F8):

```
process(clk50)
  begin
    if rising_edge(clk50) then
      D_semisync <= D_async;
      D <= D_semisync;
    end if;
  end process;
```



4. …och dessutom är det lämpligt att visa några av signaler på lysdioderna (för test och felsökning).

Veckosammanfattning

Efter kursvecka 3–4 behärskar ni…

- A/D-omvandling
  - Teori och principer
  - Successiv approximation, SAR
  - Flash och dual-slope
  - Sample-and-hold
- Tillståndsmaskiner
  - Tillståndsdiagram
  - Moore och Mealy

- Seriell kommunikation
  - RS-232
  - Principer för mottagarsynkronisering
  - Klockåtervinning
  - Serie-till-parallell-omvandling
- VHDL-implementation
  - Implementation av SAR
  - Implementation av tillståndsdiagram
  - Implementation av seriell mottagare
  - Synkroniseringsproblem: metastabilitet och otillåtna tillstånd

Kursinfo

- F10 (26 sept) avslutar VHDL-delen. Innehåll: felsökning, demo-räkning och frågestund. Ev. repetition om önskemål finns.
- Därefter övergår kursinnehållet övergår från digital till analog konstruktion, från kodning till beräkning
- Extra lab-tillfällen (frivilliga) 28 september och 17 oktober

Läs & lös

Läs:

- Lab-PM avsnitt 4
- Bengtsson avsnitt 4.4 (sample/hold)

Lös:

- Förberedelseuppgifter till lab 4, inkl 4.4–4.5

Föreläsning 10
FET och audioteknik

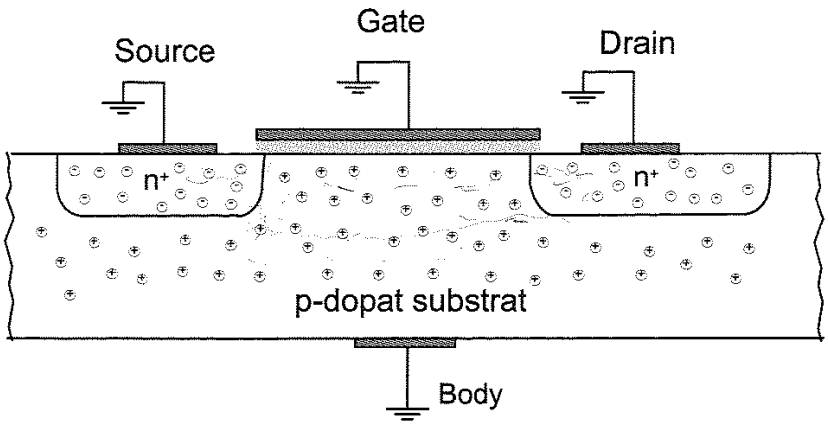
Erik Agrell 2017-09-26

Innehåll:

1. Fälteffekttransistorer
2. Ljud och decibel
3. Hörlurar och mikrofoner

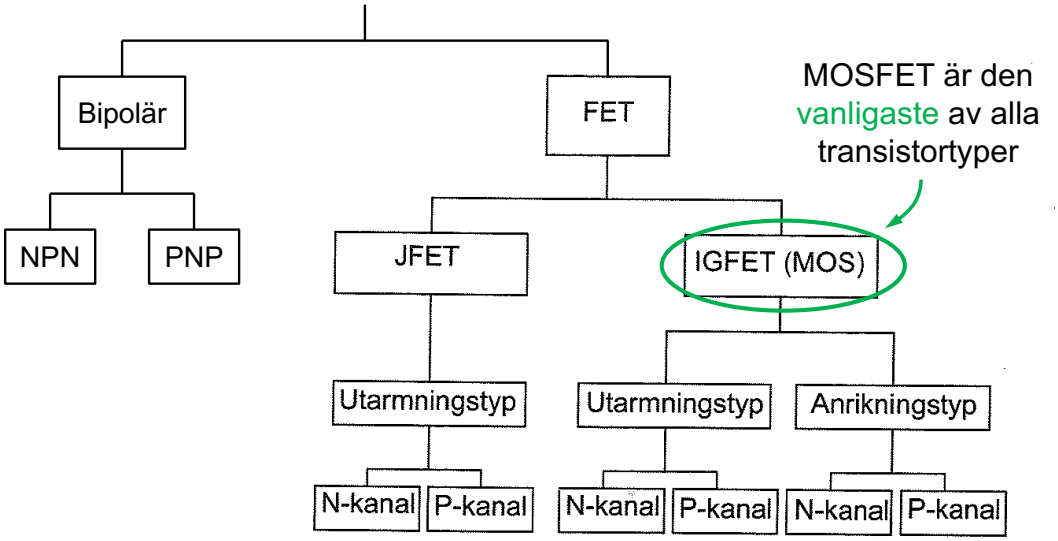
Bilder av Bengt Molin och Erik Agrell

1. Fälteffekttransistorer



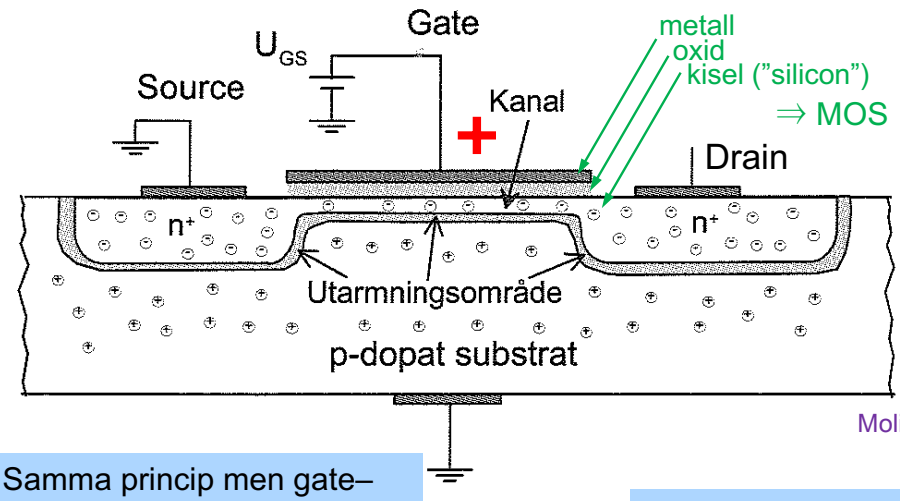
[Molin s 196]

Typer av transistorer



Från Molin s 194, utvidgad

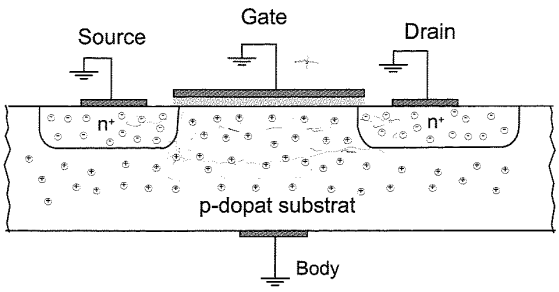
En spänning mellan gate och substrat skapar en n-dopad kanal mellan drain och source (n-kanal!)

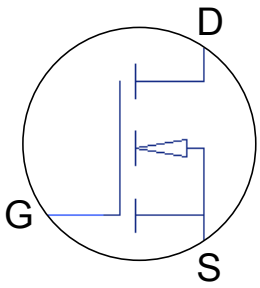


Molin s 196

JFET: Samma princip men gate–substrat består av en backspänd P-N-övergång i stället för oxidskiktet

P-kanal: byt plats på n- och p-dopning

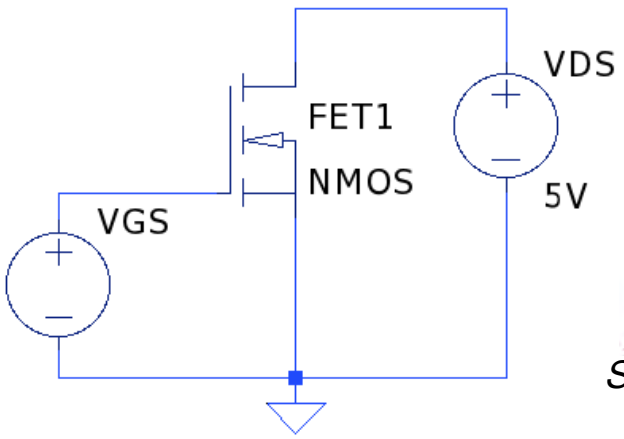




- Koppla ihop substrat och source
- Välj spänningar U_{GS} och U_{DS}
- Mät/beräkna I_D
- Men I_G då?

Gaten är isolerad (vid måttliga frekvenser)

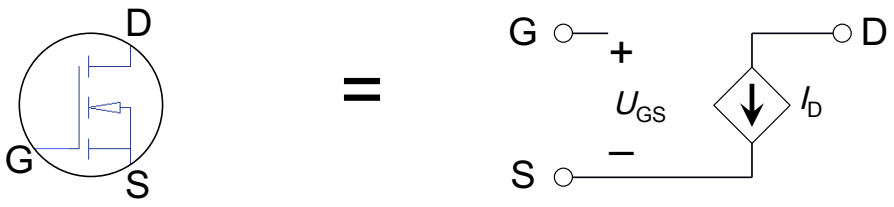
$\Rightarrow I_G = 0$



Spice demo

Slutsats: I_D är oberoende av V_{DS}, så länge som V_{DS} är tillräckligt hög

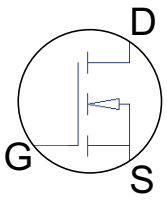
Ekvivalent schema



En MOSFET är en spänningsstyrd strömkälla

Arbetsområden och ekvationer

(n-kanal)

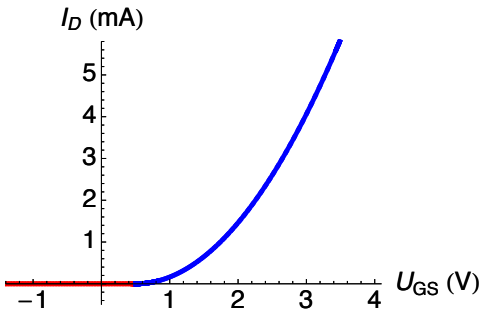


1. $U_{GS} \leq U_T$, ”strypt”:

   $I_D = 0$

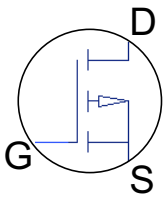
2. $0 \leq U_{GS} - U_T \leq U_{DS}$, ”mättad”:

   $I_D = \beta(U_{GS} - U_T)^2$



- Förstärkarkopplingar arbetar i det mättade området
- Där är I_D oberoende av U_{DS}
- Det finns fler områden (det linjära och genombrottsområdet), men de är inte lika viktiga

Arbetsområden och ekvationer
(p-kanal)

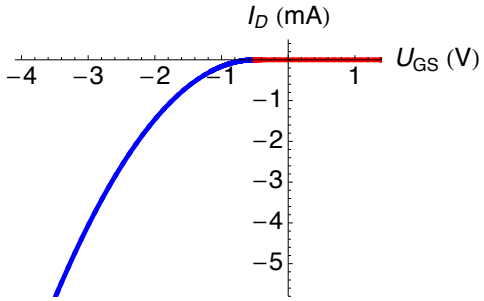


1. $-U_{GS} \le -U_T$, ”strypt”:

   $I_D = 0$

2. $0 \le -U_{GS} + U_T \le -U_{DS}$, ”mättad”:

   $-I_D = \beta(-U_{GS} + U_T)^2$



- För p-kanal gäller samma ekvationer som för n-kanal, fast med omvända tecken på alla strömmar och spänningar

Två parametrar

Tröskelspänning U_T:

- Den spänning U_{GS} som måste överskridas för att transistorn skall leda
- Kallas även pinch-off-spänning U_P
- $U_T \ge 0$ för anrikningstyp (de flesta MOS-transistorer)
- $U_T < 0$ för utarmningstyp (alla JFET)

gäller n-kanal

Koefficienten β:

- Bestämmer förstärkningen
- Om $U_T \neq 0$ används ofta

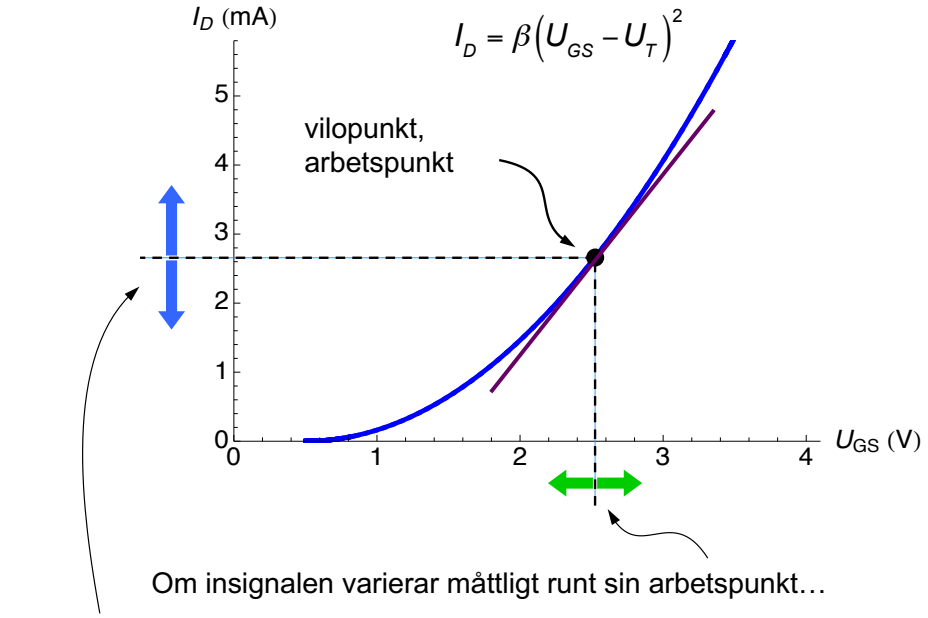
$$\beta = \frac{I_{DSS}}{U_T^2}$$

- Molin (kap. 8) använder

$$\beta = \frac{k'W}{2L}$$

där W och L är kanalens bredd resp. längd

Arbetspunkt och linearisering



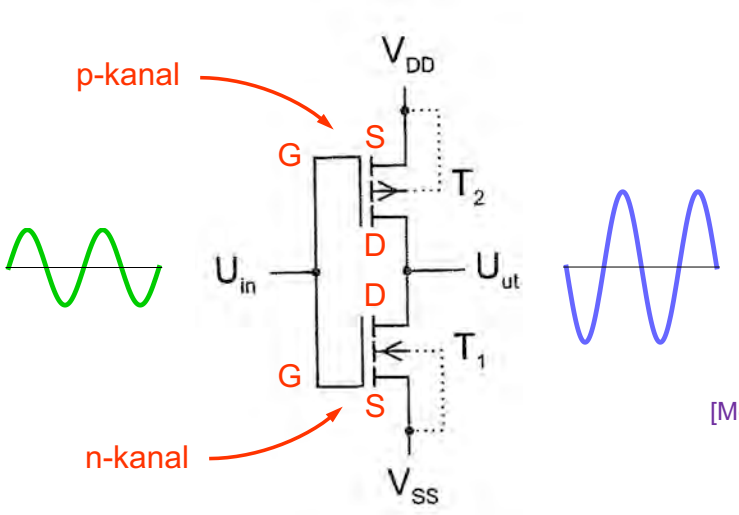
Om insignalen varierar måttligt runt sin arbetspunkt…

… så varierar utsignalen proportionellt (linjärt)

CMOS-teknik

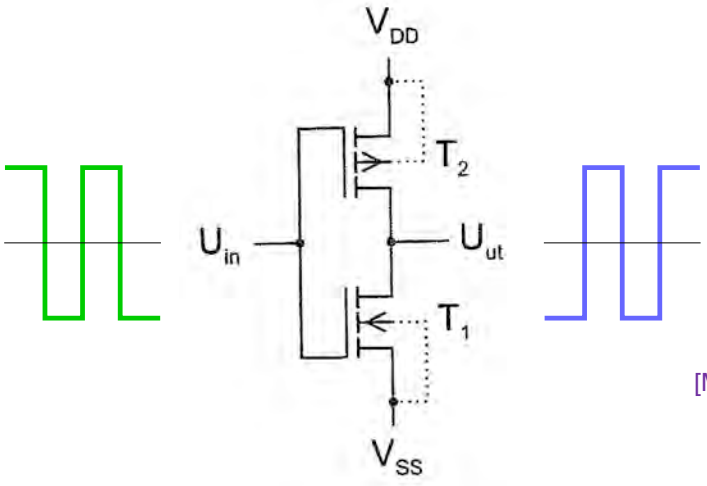
- CMOS = complementary MOS, innebär att n-kanals och p-kanals MOSFETar kombineras
- Dominerande teknik i integrerade kretsar
- Mycket energisnål

Exempel 1: CMOS-förstärkare



[Molin s 352]

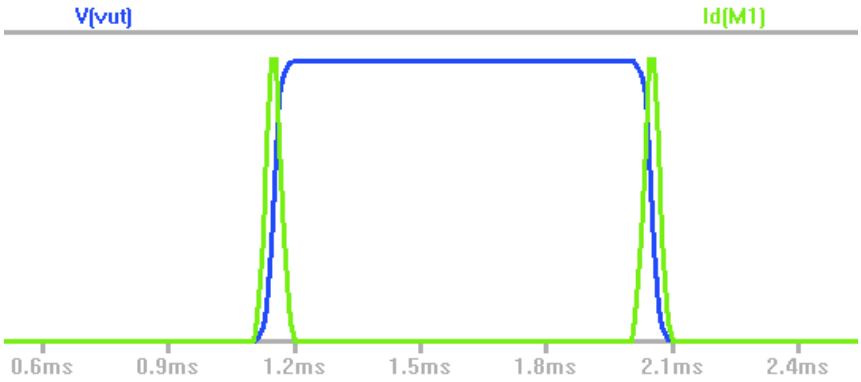
Exempel 2: CMOS-inverterare



[Molin s 352]

En inverterare är en inverterande förstärkare med binär (maximalt utstyrd) insignal

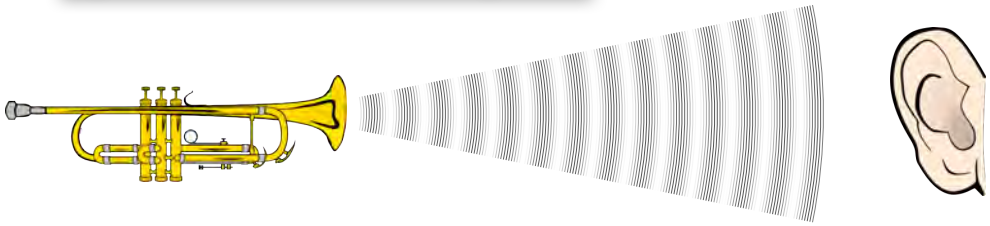
Energiförbrukning i CMOS



- CMOS-kretsar drar bara ström vid omslag
- Energiförbrukningen i processorer och minnen ökar vid belastning
- Energiförbrukningen är proportionell mot klockfrekvensen

2. Ljud och decibel

Ljud är variationer i lufttryck…



…som får trumhinnan att vibrera

Upplevd ljudnivå

Ljudtrycksnivå:

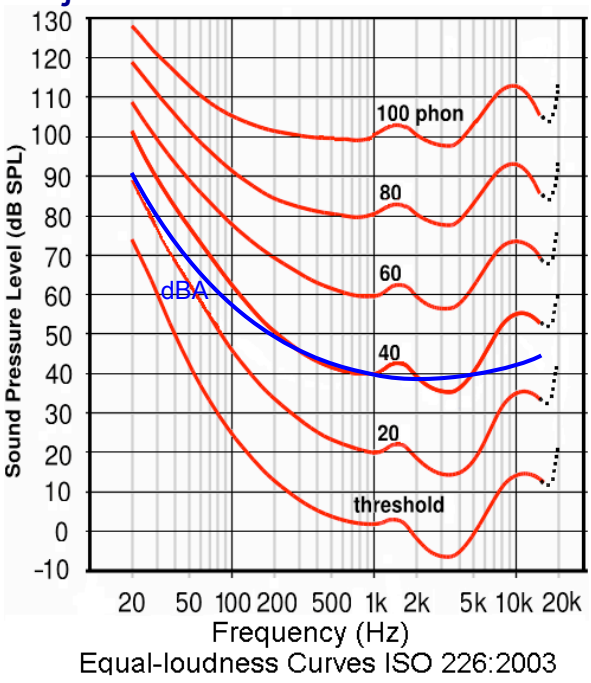
$$\text{dB SPL} = 20\log\frac{p}{p_0}$$

där

p = ljudtryck

$p_0 = 20\ \mu\text{Pa}$

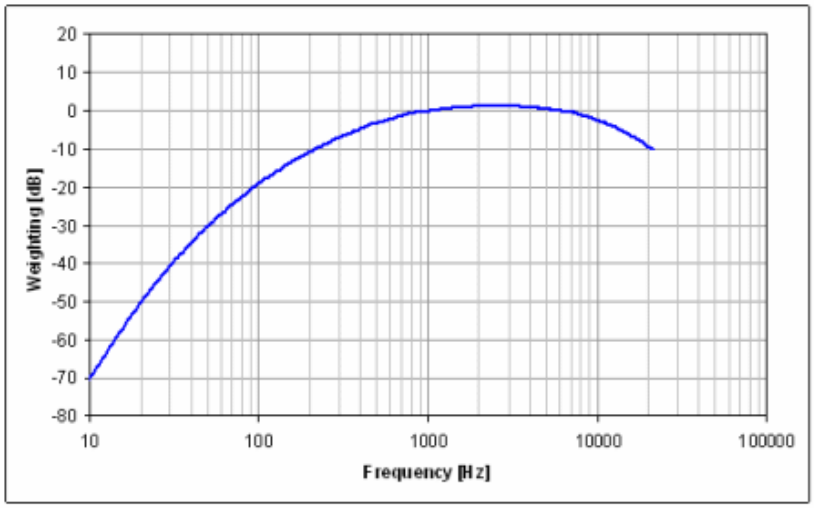
Samma ljudtrycksnivå uppfattas olika stark beroende på frekvensen



Equal-loudness Curves ISO 226:2003

[diracdelta.co.uk, http://www.sengpielaudio.com]

Frekvensberoende viktning



[diracdelta.co.uk]

Upplevd ljudnivå (dBA) = ljudtrycksnivå (dB SPL) + viktning (dB)

Exempel: 70 dB SPL vid 100 Hz upplevs som 50 dBA

Noise Source	Level (dBA)
Jet takeoff from 25m (possible eardrum damage)	150
Aircraft carrier flight deck	140
Jet take-off from 100m	130
Rock band	110–120
Jet fly-by at 300m, car horn at 1m	100–110
Petrol grass mower - operator	90–100
Food blender - operator, busy urban street	80–90
Accelerating vehicle from 50kmhr-1 at 7.5m	70–80
Vacuum cleaner - operator	60–70
Light traffic at 30m	50–60
Quiet residential area - daytime	40–50
Quiet residential area - nighttime	30–40
Wilderness, rustling leaves, whisper	20–30
Breathing	10
Threshold of human hearing	0

[diracdelta.co.uk]

Socialstyrelsens författningssamling

Ansvarig utgivare: Chefsjurist Nils Blom

Socialstyrelsen

Socialstyrelsens allmänna råd om höga ljudnivåer;

SOSFS 2005:7 (M)

beslutade den 15 mars 2005.

Utkom från trycket den 15 april 2005

Tabell 1 – Riktvärden för lokaler och platser dit barn under 13 års ålder inte har tillträde

Maximalt ljud	L_{AFmax}[1]	115 dB
Ekvivalent ljud	L_{AeqT}[2]	100 dB

1 Den högsta A-vägda ljudnivån.
2 Den A-vägda ekvivalenta ljudnivån under en viss tidsperiod (T).

Tabell 2 – Riktvärden för lokaler och platser dit både barn och vuxna har tillträde

Maximalt ljud	L_{AFmax}[1]	110 dB
Ekvivalent ljud	L_{AeqT}[2]	97 dB[3]

1 Den högsta A-vägda ljudnivån.
2 Den A-vägda ekvivalenta ljudnivån under en viss tidsperiod (T).
3 Särskild hänsyn bör tas i verksamheter som är särskilt riktade till barn, s.k. knattediskotek eller liknande. Där bör ekvivalenta A-vägda ljudnivåer under 90 dB alltid eftersträvas.

Fler decibelstorheter

Definition

Effektkvot $\quad dB = 10\log\frac{P_1}{P_0}$

Härledda storheter

Energikvot $\quad dB = 10\log\frac{E_1}{E_0}$

Spänningskvot $\quad dB = 20\log\frac{U_1}{U_0}$

Strömkvot $\quad dB = 20\log\frac{I_1}{I_0}$

Speciella referensnivåer

Effekt $\quad dBW = 10\log\frac{P}{1W}$

Effekt $\quad dBm = 10\log\frac{P}{1mW}$

Spänning $\quad dBV = 20\log\frac{U_{eff}}{1V}$ Obs!

Ljudtryck $\quad dB\ SPL = 20\log\frac{p}{20\mu Pa}$

Den magiska dB-regeln

1 dB förändring någonstans

⇒

1 dB (dBW, dBm, dBV, dB SPL, dBA,…) förändring överallt

… i alla linjära system

Två vanliga dB-misstag

- Decibelvärden skall aldrig multipliceras eller divideras!
- Det finns inget som heter ”effekt-dB” eller ”spännings-dB”. Det blir samma dB oavsett om man beräknar från effekt eller spänning!

3. Hörlurar och mikrofoner

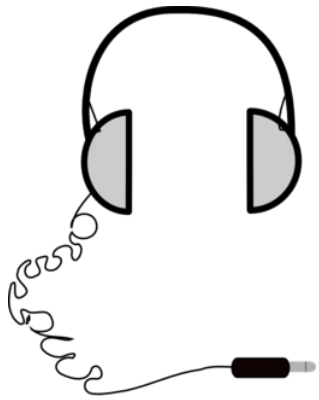
Hörlurar:

- En hörlur omvandlar ström till ljud
- Ljudtrycket är proportionellt mot strömmen
- I datablad anges hörlurars känslighet ofta i dB SPL @ 1 mW (t.ex. 100 dB SPL @ 1 mW)
- Hörlurens impedans är typiskt låg, några 10-tals Ω ⇒ relativt hög ström

Skrivs ibland dB SPL/mW, vilket är något missvisande

Exempel
(nyttigt för lab 5)

- Vilken spänning krävs för att en hörlur med impedans 32 Ω och känslighet 90 dB SPL @ 1 mW skall producera en ljudtrycksnivå på 106 dB SPL?



□

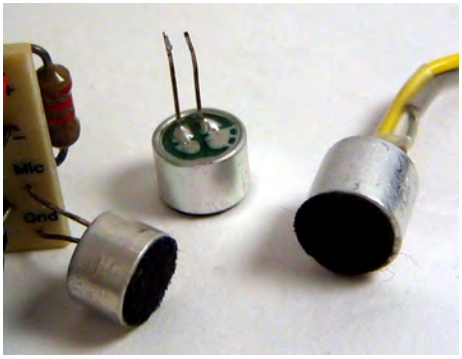
Svar: 1.13 V effekttivvärde (3.20 V_{pp})

Mikrofoner:

- En mikrofon omvandlar ljud till ström
- Strömmen är proportionellt mot ljudtrycket
- I datablad anges ofta mikrofoners känslighet i dBV @ 1 Pa vid en viss inimpedans (t.ex. –39 dBV @ 1 Pa vid 2 kΩ impedans)
- Begreppet inimpedans syftar här inte på mikrofonen utan på kretsen som mikrofonen matar

Skrivs ibland dBV/Pa, vilket är något missvisande

Elektretmikrofon



En elektretmikrofon fungerar som en spänningskälla + JFET

d.v.s. som en växelströmkälla

En elektretmikrofon har två ben, som ger utsignalen.

Men var lägger man då matningen?

Svar: Mellan samma två ben som utsignalen. Inga problem: Matningen är DC, utsignalen är AC!

… fortsättning följer i F11…

Läs & lös

Läs:

- Molin s 193–198, 206–210, 228 och 352 om FET
- Molin s 23–26 om decibel

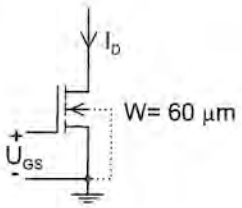
Lös:

- Molin uppg 8.2
- Molin uppg 9.2 men med $\lambda=0$
- Inlämningsuppgift 3

8.2 a Transistorn i vidstående koppling sitter i en koppling som ger strömmen $I_D = 0{,}5$ mA. Vad blir U_{GS}?

Antag kanallängd $L = 4$ µm och $k' = 150\ \mu A/V^2$ $U_T = 0{,}7$ V

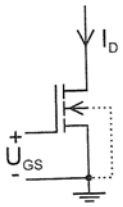
b Samma transistor sitter nu i en koppling som ger spänningen $U_{GS} = 1{,}0$ V till transistorn. Vad blir I_D?



9.1 Vidstående NMOS-transistor har $L = 4$ µm och $W = 100$ µm. Beräkna g_m för följande värden på U_{GS}: 1,0 V , 1,2 V och 1,4 V.

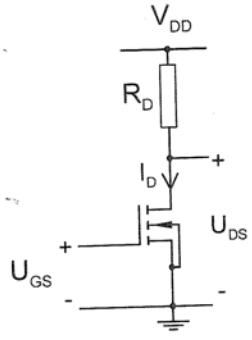
Transistorparametrar:

$k' = 200\ \mu A/V^2$ $U_T = 0{,}8$ V ~~$\lambda = 0{,}01$~~ $\lambda = 0$



9.2 Använd transistorn i föregående uppgift och dimensionera R_D så att U_{DS} blir 1,5 V med $U_{GS} = 1{,}0$ V. $V_{DD} = 3$ V.

Hur stor variation blir det på U_{DS} om U_{GS} varierar ± 10 mV?



Föreläsning 11
Förstärkare och deras frekvensberoende

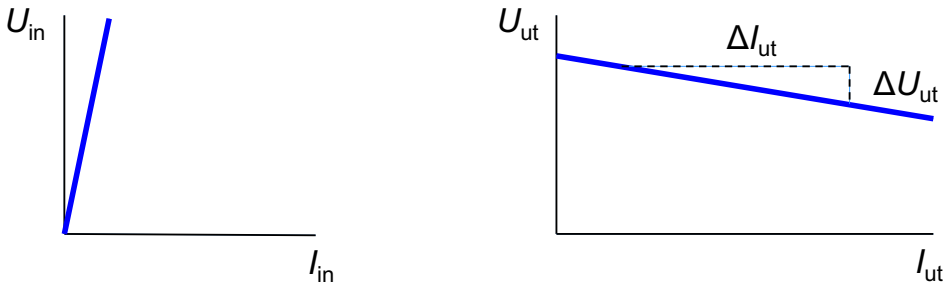
Erik Agrell 2017-09-27

Innehåll:

1. In- och utimpedans
2. Mer om mikrofoner
3. Två sorters förstärkare
4. Bandbreddsbegränsning

Bilder av Bengt Molin och Erik Agrell

1. In- och utimpedans

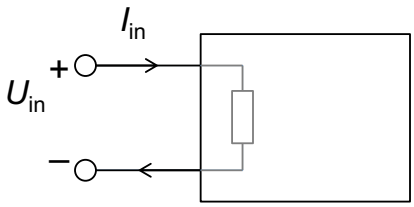


Inimpedansen definieras som

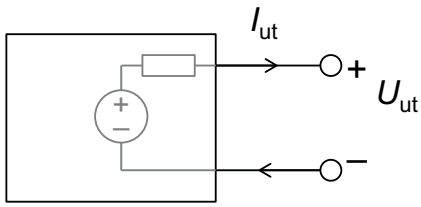
$$Z_{in} = \frac{U_{in}}{I_{in}}$$

Utimpedansen definieras som

$$Z_{ut} = \frac{\Delta U_{ut}}{\Delta I_{ut}}$$



$$Z_{in} = \frac{U_{in}}{I_{in}}$$



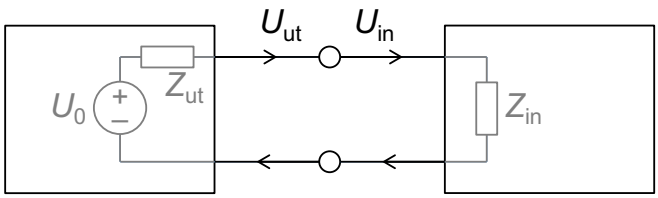
$$Z_{ut} = \frac{\Delta U_{ut}}{\Delta I_{ut}}$$

Man vill ha:

- hög Z_{in} $\Rightarrow$ kretsen drar ingen ström på ingången
- låg Z_{ut} $\Rightarrow$ spänningen sjunker inte även om man belastar utgången (tar ut ström)

Undantag: I högfrekvenssystem behöver man $Z_{ut} = Z_{in}$ för att undvika reflektioner

Ihopkopplad in- och utimpedans



Om man kopplar ihop två kretsar blir det spänningsdelning mellan Z_{ut} och Z_{in}

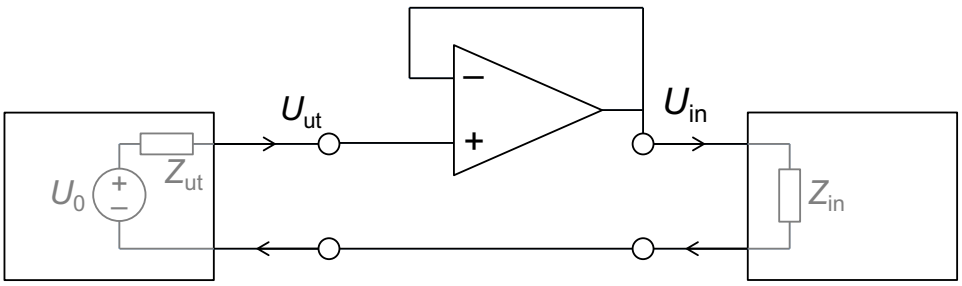
$$U_{in} = U_0 \frac{Z_{in}}{Z_{in} + Z_{ut}}$$

Om Z_{ut} är mycket mindre än Z_{in}:

$$U_{in} \approx U_0$$ **OK**

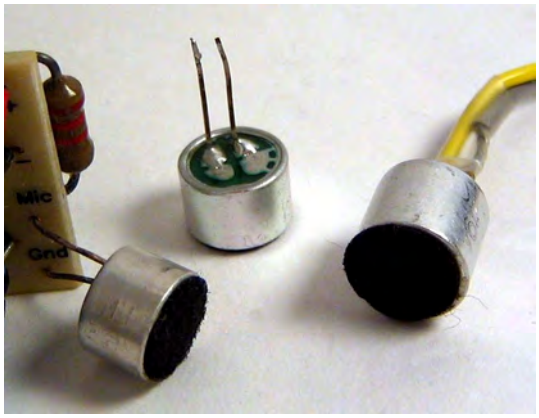
Men om Z_{ut} *inte* är mycket mindre än Z_{in} då?

Svar: Då behövs en spänningsföljare.



Nu är $U_{in} = U_{ut} = U_0$ oavsett Z_{ut} och Z_{in}

2. Mer om mikrofoner



(från F10)

En elektretmikrofon fungerar som en spänningskälla + JFET

d.v.s. som en växelströmkälla

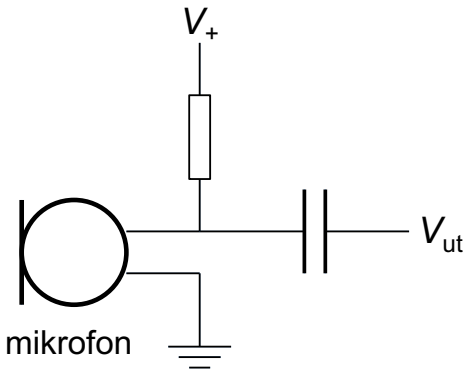
En elektretmikrofon har två ben, som ger utsignalen.

Men var lägger man då matningen?

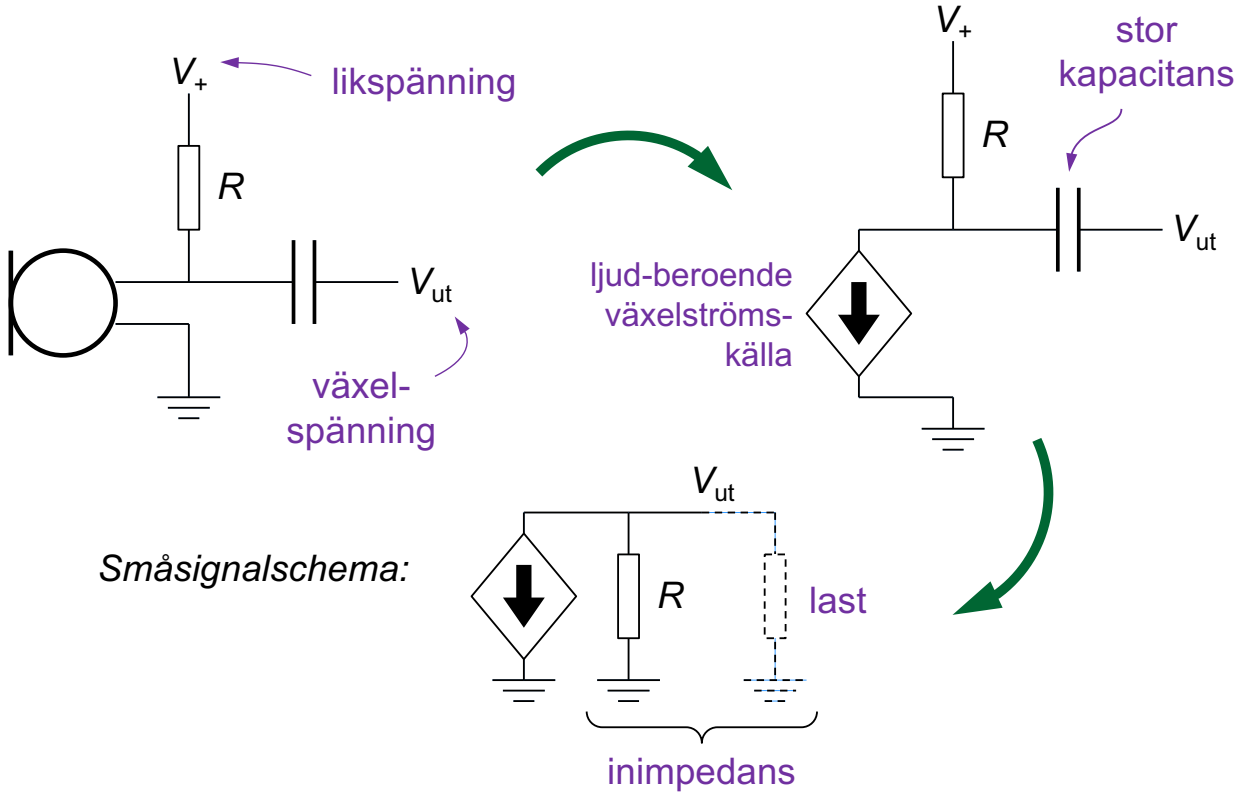
Svar: Mellan samma två ben som utsignalen. Inga problem: Matningen är DC, utsignalen är AC!

Inkoppling av mikrofonen

- Matningen (DC) läggs mellan samma två ben som ger utsignalen (AC)
- Matningen måste därför gå via en resistor
- Utsignalen måste gå via en kondensator



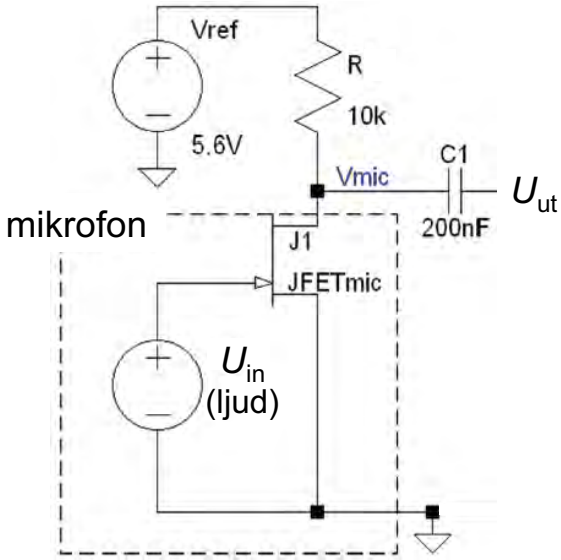
Ekvivalenta scheman



Exempel: elektretmikrofon

(nyttig inför lab 5)

- Enligt datablad har en viss elektretmikrofon känslighet –40 dBV @ 1Pa vid en inimpedans på 5 kΩ. Vilken känslighet har den i nedanstående koppling?



Utsignalens styrka beror på R

□

Svar: –34 dBV @ 1 Pa

3. Två sorters förstärkare

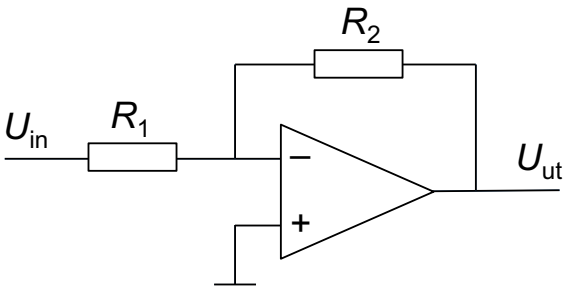
Fall 1: Spänningen räcker inte till

Svag signalkälla (ger ofta några mV)	→	Signal-förstärkare	→	Elektronik (behöver ofta >1V)
Givare, mätinstrument, mikrofon, …		Ungefär samma ström (liten)		Dator, digital signalbehandling (DSP), A/D, …

Signalförstärkare byggs ofta med operationsförstärkare

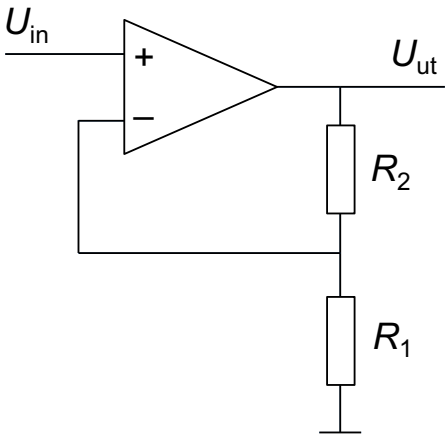
Signalförstärkare

Inverterande förstärkare:



$$A = \frac{U_{ut}}{U_{in}} = -\frac{R_2}{R_1}$$

Icke inverterande förstärkare:



$$A = \frac{U_{ut}}{U_{in}} = \frac{R_1 + R_2}{R_1}$$

Fall 2: Strömmen räcker inte till

Elektronik (ger ofta <10 mA) → Effekt-förstärkare → Elpryl (behöver ofta >0.1 A)

Dator, digital signalbehandling (DSP), D/A, …

Ungefär samma spänning (några V)

Motor, lampa, display, högtalare, pump, värmare, …

Effektförstärkare byggs ofta med transistorer

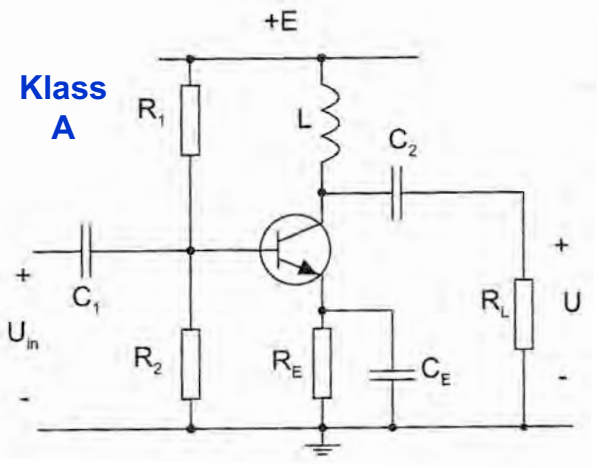
Effektförstärkare

Typer av effektförstärkare:

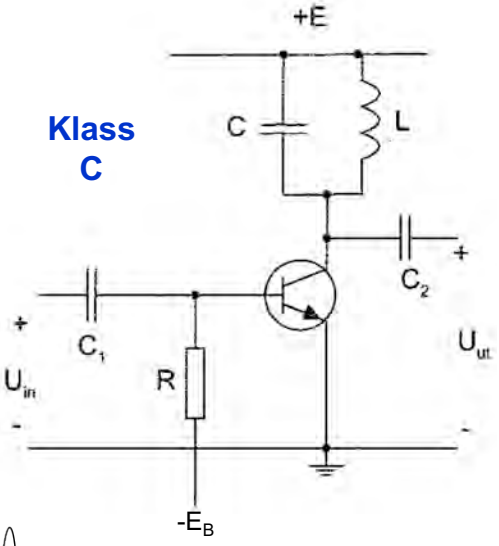
- Klass A: en transistor leder hela tiden
- Klass B: två transistorer leder halva tiden var
- Klass C: en transistor leder en mindre del av tiden och en svängningskrets snyggar till utsignalen
- Klass D: digital logik och pulsbreddsmodulering
- Klass AB: två transistorer leder lite mer än halva tiden var

Exempel på klass A, C och D

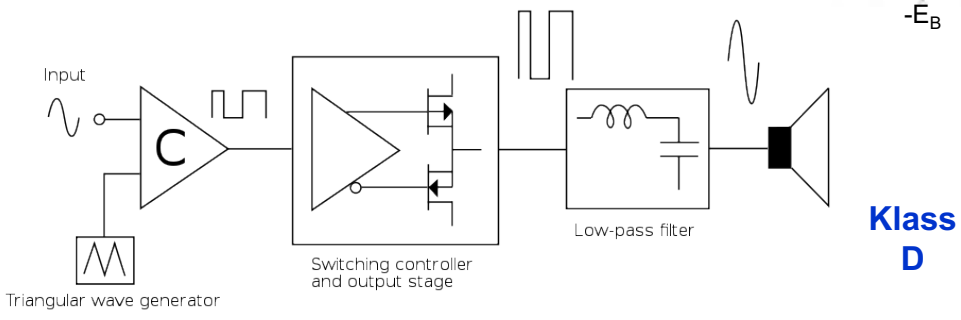
Klass A



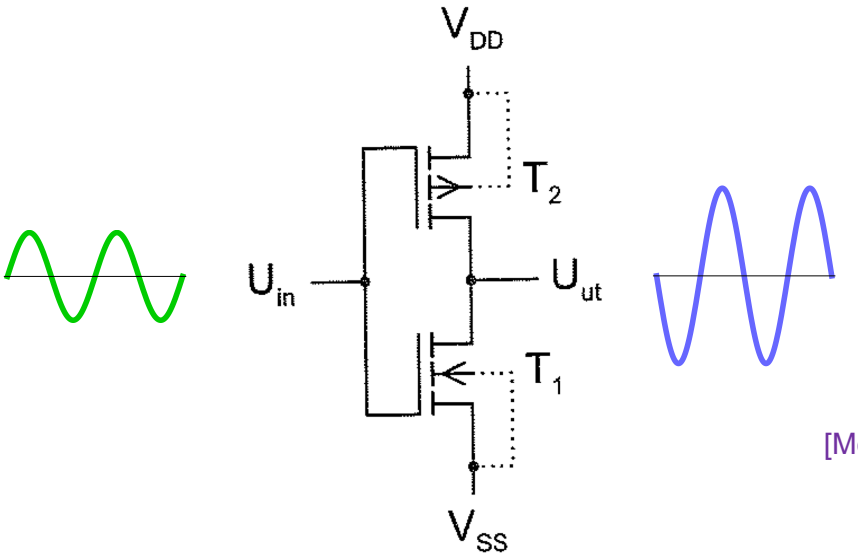
Klass C



Klass D

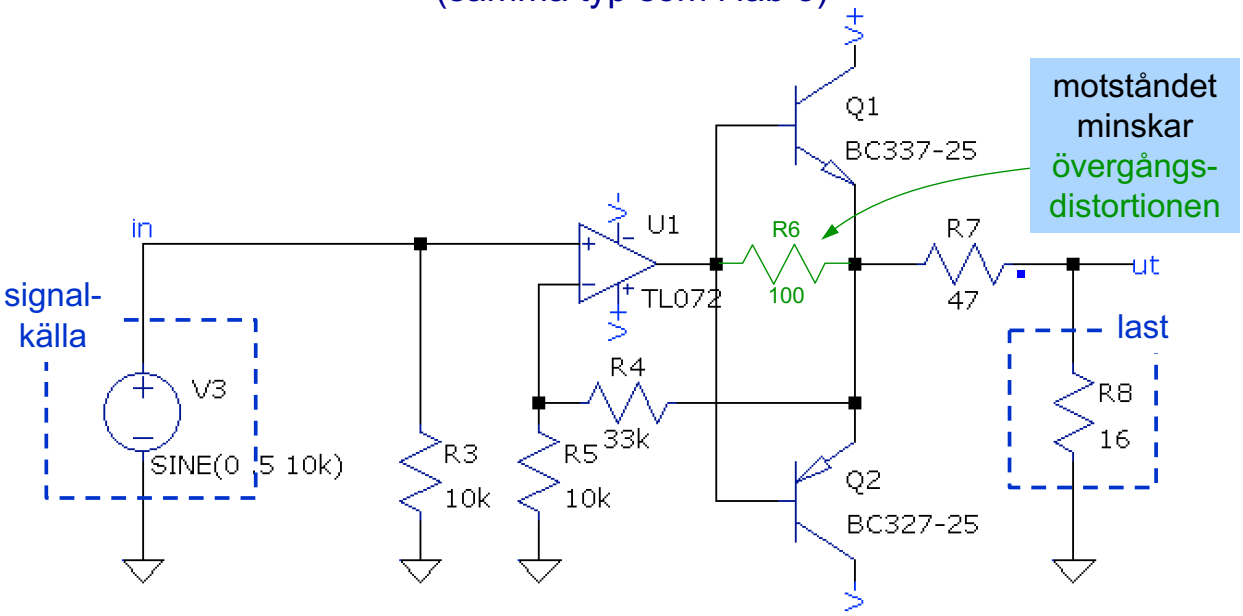


Exempel på klass B
(CMOS-förstärkaren från F10)



[Molin s 352]

Exempel på klass B
(samma typ som i lab 5)



Spice demo

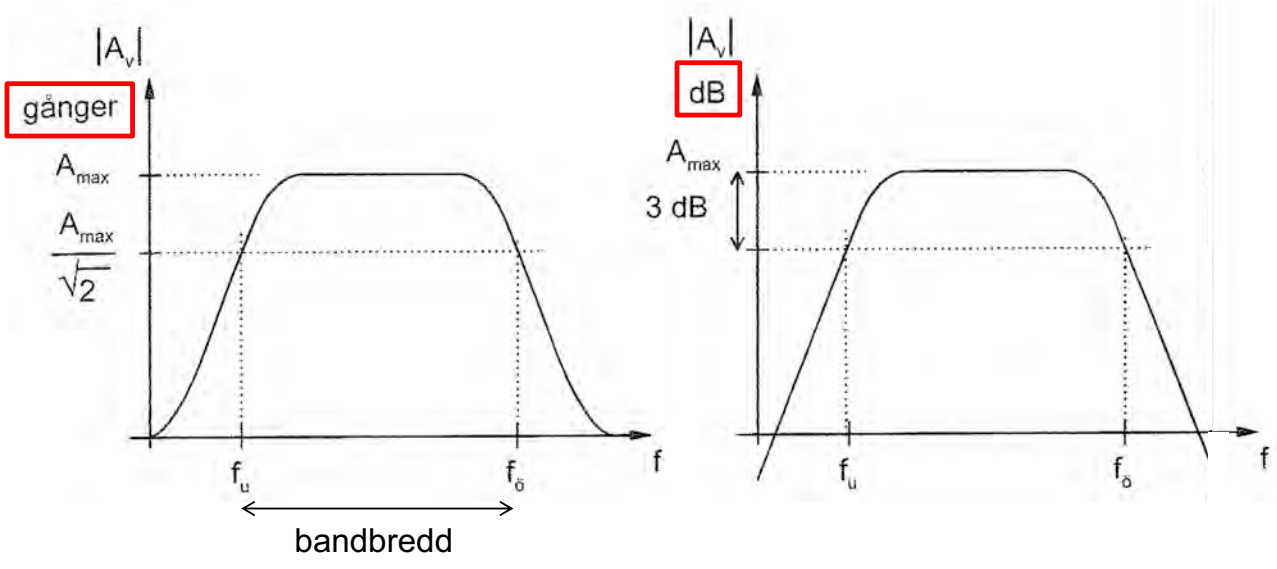
4. Bandbreddsbegränsning

En förstärkare bör förstärka de önskade signalerna, som ligger inom ett visst frekvensområde:

- audiosignaler: 20 Hz–10 kHz
- radiosignaler: ett smalt spektrum runt bärfrekvensen

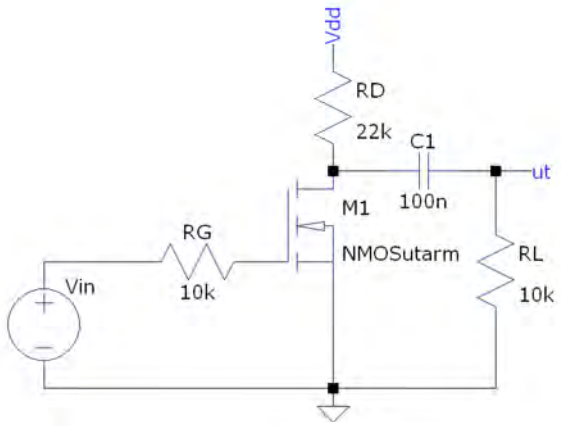
Samtidigt vill man dämpa allt utanför det önskade frekvensområdet (störningar)

Gränsfrekvenser och bandbredd

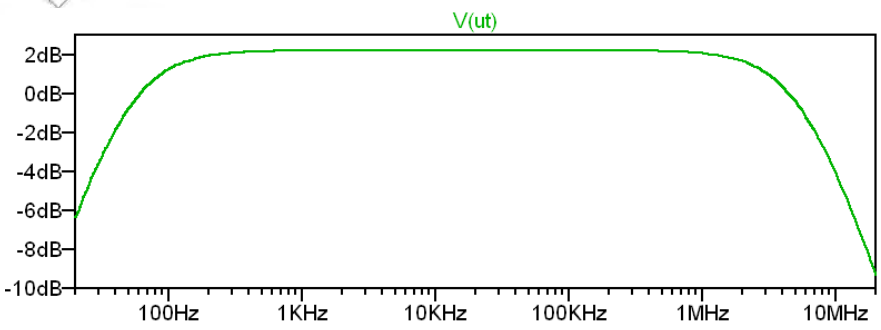


[Molin s 31]

Exempel 1: FET-förstärkare klass A



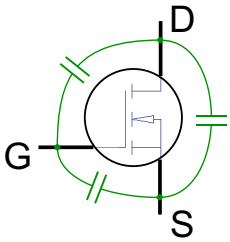
Förstärkningen minskar både vid låga och höga frekvenser – varför?



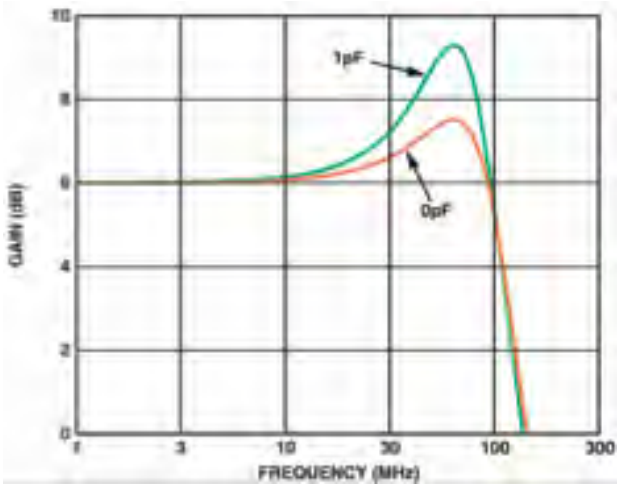
Varför minskar förstärkningen?

Vid låga frekvenser: Kondensatorn på utgången spärrar.

Vid höga frekvenser: ”parasitkapacitanser”



- Alla kretsar innehåller små kapacitanser mellan anslutningarna
- Impedansen minskar när frekvensen ökar: $Z=1/(j\omega C)$
- Vid mycket höga frekvenser är kretsen i princip kortsluten
- Högfrekvenselektronik kräver därför speciella konstruktionsmetoder



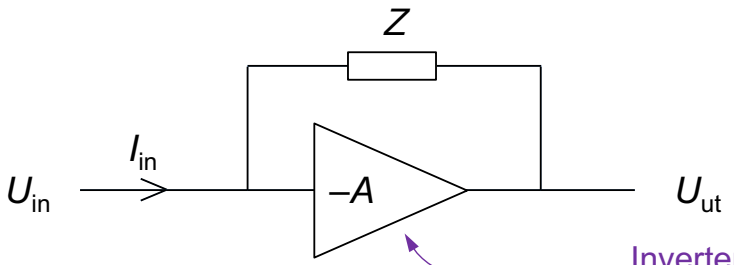
ELEKTRONIK TIDNINGEN

*”**I höghastighetskretsar** krävs det inte mycket för att parasiterna ska påverka kretsens prestanda. Det kan räcka med en förändring på några tiondels pikofarad. För att ge ett exempel: en enda pF extra parasit-kapacitans vid förstärkarens inverterande ingång kan ge ett nästan 2 dB högre toppvärde i frekvensdomänen (Figur 3).”*

[John Ardizzoni, Elektroniktidningen 2006]

Millereffekten

Antag att det finns en impedans (vanligen kapacitiv) mellan en förstärkares in- och utgång:

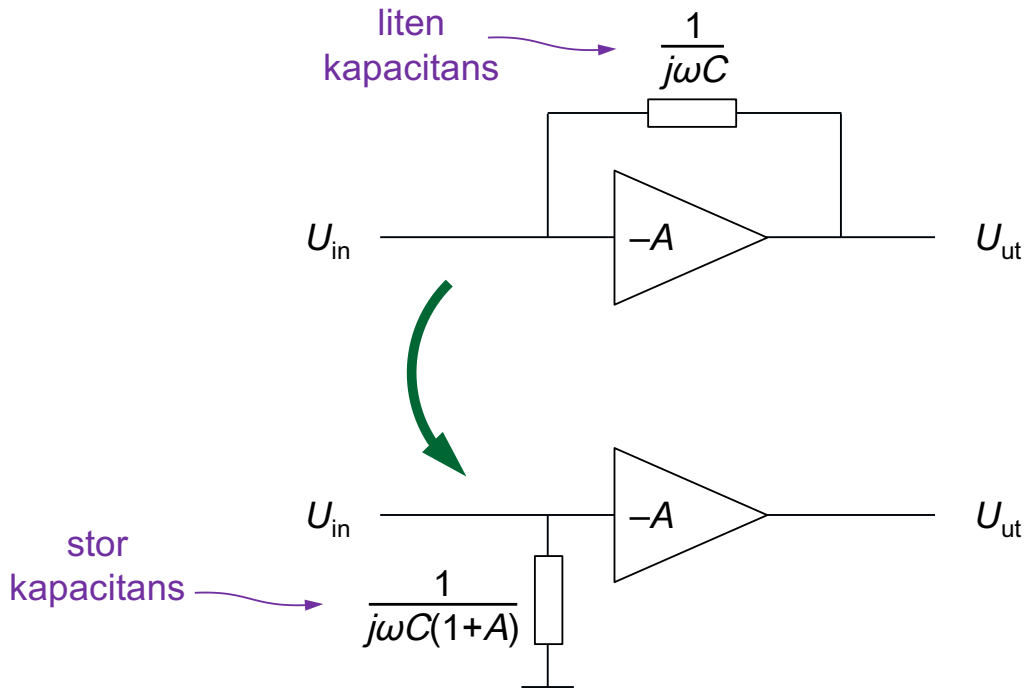


Inverterande spänningsförstärkare med förstärkning –*A*, t.ex. återkopplad OP

Hur stor är inimpedansen? □

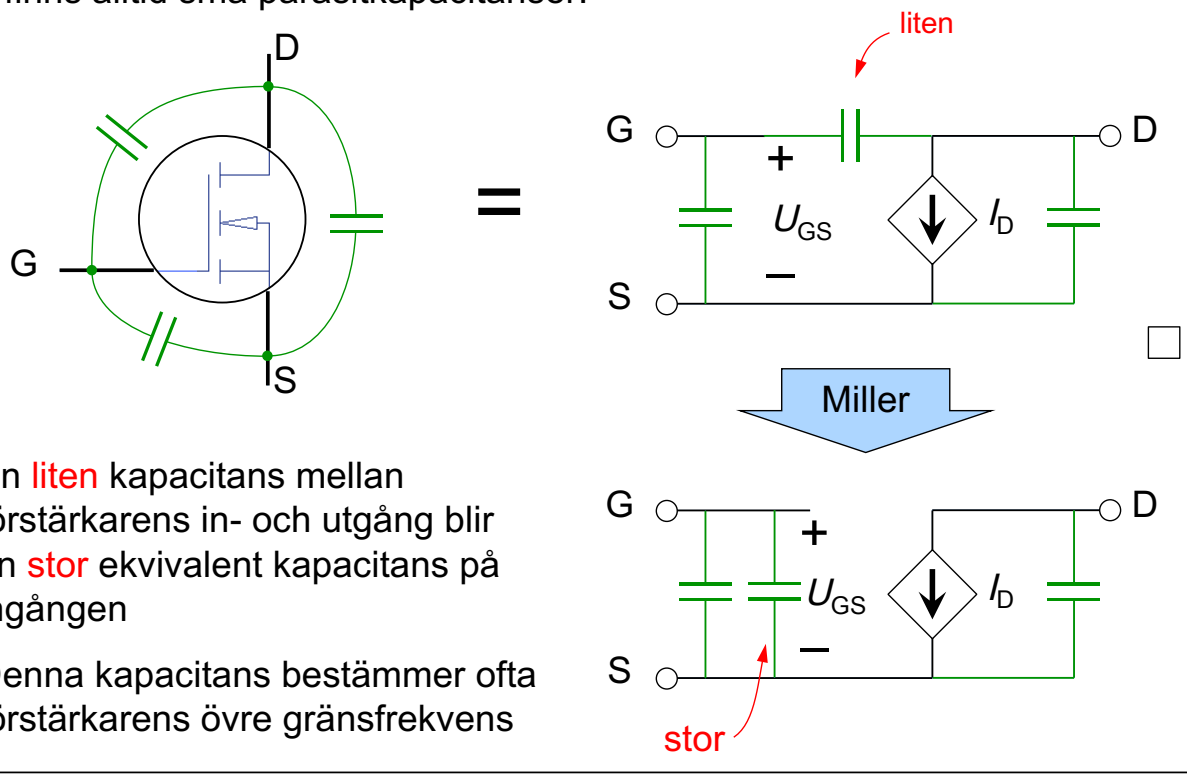
Svar: $\frac{Z}{1+A}$

Ekvivalent schema



Millerkapacitanser

Det finns alltid små parasitkapacitanser:



- En liten kapacitans mellan förstärkarens in- och utgång blir en stor ekvivalent kapacitans på ingången
- Denna kapacitans bestämmer ofta förstärkarens övre gränsfrekvens

Veckosammanfattning

Efter kursvecka 5 behärskar ni…

- Fälteffekttransistor
  - Ekvivalent schema
  - Strypt och mättat område
  - n-kanal och p-kanal
  - Förstärkare och inverterare i CMOS-teknik
- Ljud och decibel
  - Ljudtrycksnivå i dB SPL
  - dBA-viktning
  - Samband mellan dB och effekt- eller spänningskvot
  - dBm, dBW och dBV
  - Den Magiska Decibel-regeln

- Hörlurar och mikrofoner
  - Känslighet
  - Inkoppling av elektretmikrofon
- In- och utimpedans
  - Definitioner
  - Problem vid ihopkoppling
- Effektförstärkare
  - Typer
  - Analys av klass B
  - Övergångsdistortion
- Bandbegränsingar i elektronik
  - Gränsfrekvenser och bandbredd
  - Parasitkapacitanser
  - Millereffekten

Läs & lös

Läs:

- Molin s 23–32 (förstärkarprinciper)
- Molin s 45–51 och 54–55 (signalförstärkare)
- Molin s 400 och 408–415 (effektförstärkare)
- Molin s 73–74 (in- och utresistans, slew rate)
- Molin s 322–323, 246 (Millereffekten)

Lös:

- Inlämningsuppgift 3
- Förberedelseuppgifter till lab 5

Föreläsning 12
Analoga filter

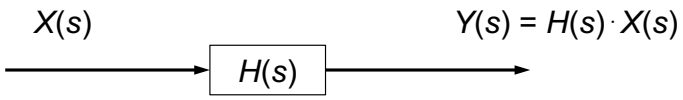
Erik Agrell 2017-10-03

Innehåll:

1. Filterteori
2. Filtertyper

Bilder av Lars Bengtsson och Erik Agrell

1. Filterteori



Överföringsfunktion $$H(s) = \frac{(1-s/n_1)\cdots(1-s/n_N)}{(1-s/p_1)\cdots(1-s/p_P)} = A\frac{(s-n_1)\cdots(s-n_N)}{(s-p_1)\cdots(s-p_P)}$$

- n_1, n_2, … är nollställen
- p_1, p_2, … är poler

$$A = \frac{p_1\cdots p_P}{n_1\cdots n_N}$$

Poler och nollställen

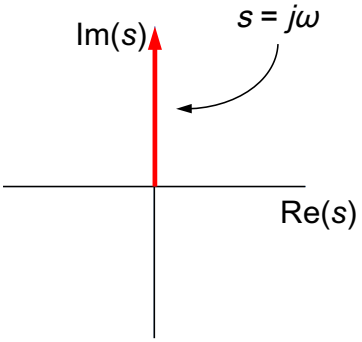
- Poler och nollställen är komplexa
- $H(s)$ har reella koefficienter

$\Rightarrow$ poler och nollställen är antingen reella eller förekommer i komplexkonjugerade par

- Stabilt filter $\Leftrightarrow$ alla poler i vänstra halvplanet

Frekvenskarakteristik

I det komplexa s-planet kan den positiva imaginäraxeln tolkas som frekvensaxeln



Om vi vet överföringsfunktionen $H(s)$ kan vi räkna ut:

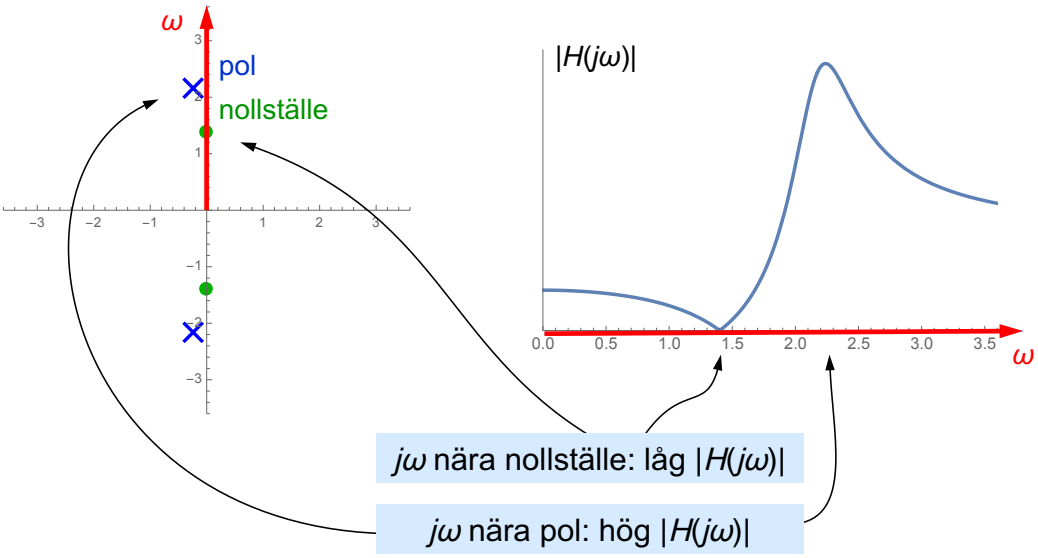
- Frekvensfunktionen $H(j\omega)$
- Amplitudfunktionen $|H(j\omega)|$
- Fasfunktionen $\arg H(j\omega) = \angle H(j\omega)$

Specialfall:

- $H(0)$ bestämmer DC-egenskaper
- $H(\infty)$ bestämmer högfrekvens-egenskaper

Amplitudfunktion

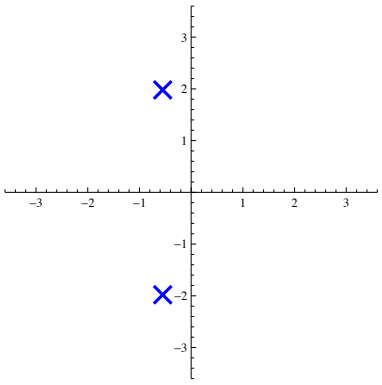
$$H(j\omega) = A\frac{(j\omega - n_1)\cdots(j\omega - n_N)}{(j\omega - p_1)\cdots(j\omega - p_P)} \quad \Rightarrow \quad |H(j\omega)| = A\frac{|j\omega - n_1|\cdots|j\omega - n_N|}{|j\omega - p_1|\cdots|j\omega - p_P|}$$

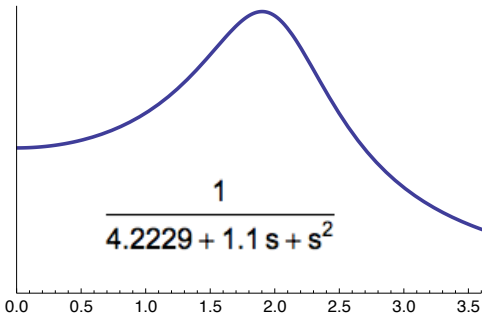


$j\omega$ nära nollställe: låg $|H(j\omega)|$

$j\omega$ nära pol: hög $|H(j\omega)|$

Filterdemonstration 1

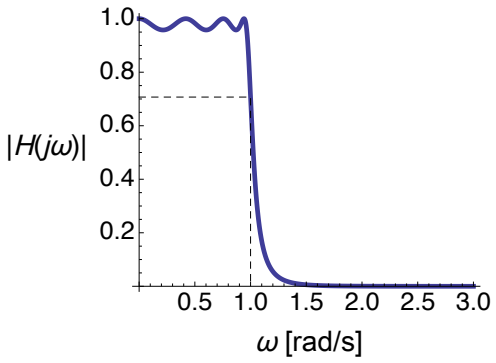




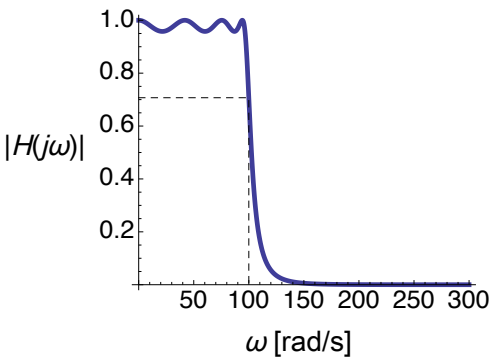
Mathematica demo

Frekvenstransformering

Vi har detta:



Vi vill ha detta:



Vad gör vi?

Svar: byt s mot $s/100$ i $H(s)$!

Om $H(s)$ har bandbredd B,

så har $H(s/c)$ bandbredd $c \cdot B$

- Motsvarande gäller gränsfrekvenser, centerfrekvenser etc., eftersom hela frekvensaxeln skalas om.
- Många filter specificeras för gränsfrekvensen 1 rad/s.

Exempel

Ett 3e ordningens LP-filter (Chebyshev) med gränsfrekvens 1 rad/s har överföringsfunktionen

$$H_1(s) = \frac{1}{1 + 3.71s + 3.38s^2 + 4s^3}$$

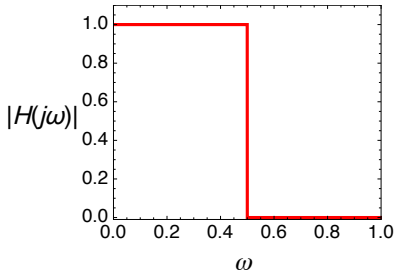
Beräkna överföringsfunktionen för ett Chebyshev-filter med gränsfrekvens 1 kHz.

□

Svar: $H(s) = \dfrac{1}{1 + 5.91 \cdot 10^{-4} s + 6.04 \cdot 10^{-8} s^2 + 1.61 \cdot 10^{-11} s^3}$

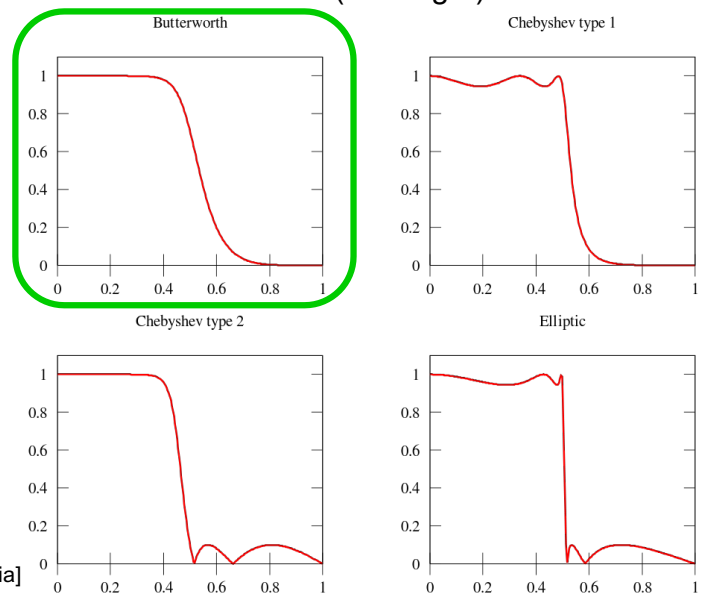
2. Filtertyper

Idealt LP-filter:



(kan inte byggas)

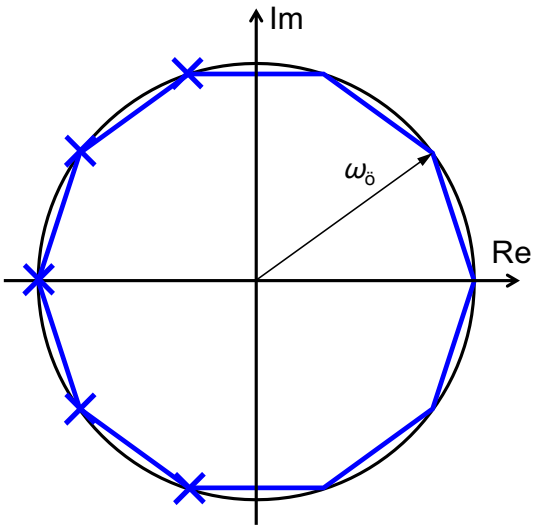
Praktiska LP-filter (ordning 5):



[Wikipedia]

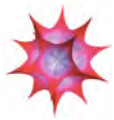
Butterworth-filter

En maximalt flat filterkarakteristik får man genom att placera polerna jämnt utspridda på en halvcirkel

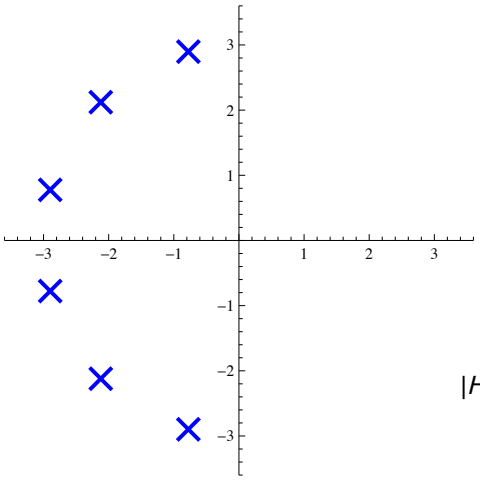


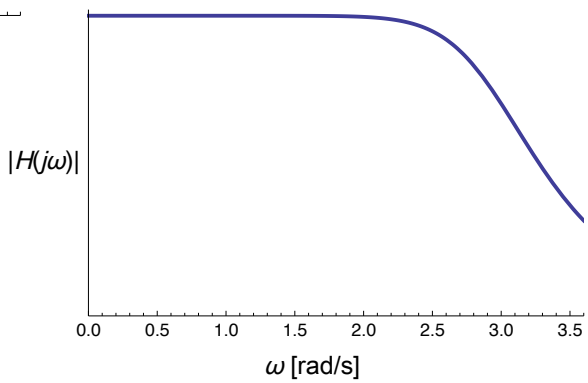
- Cirkelns radie är gränsfrekvensen i rad/s
- ”jämnt utspridda” betyder som hörnen på en 2*n*-hörning (för filterordning *n*)

Filterdemonstration 2

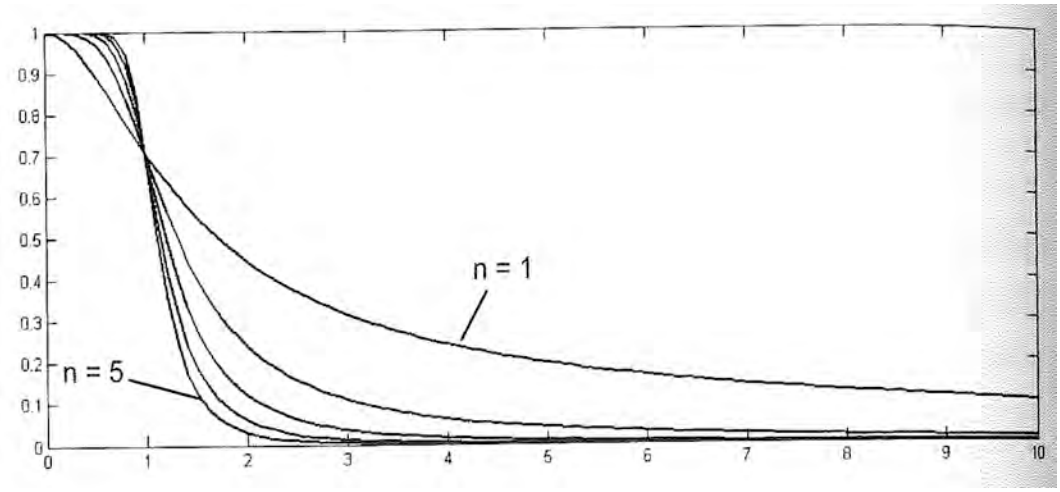


Mathematica demo



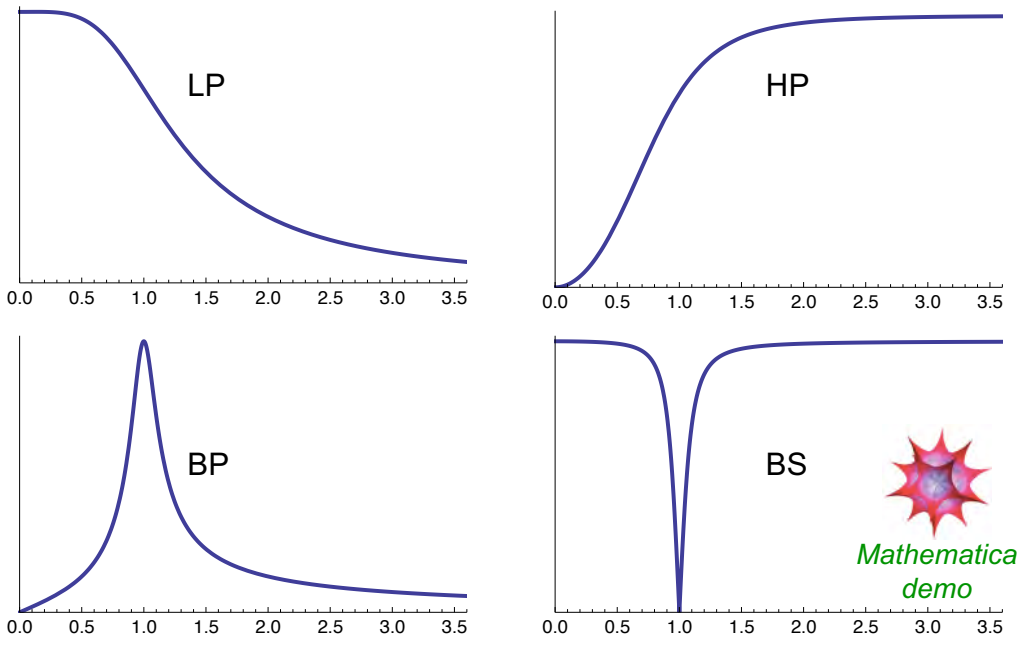


Butterworth-filter av olika ordning



[Bengtsson s 424]

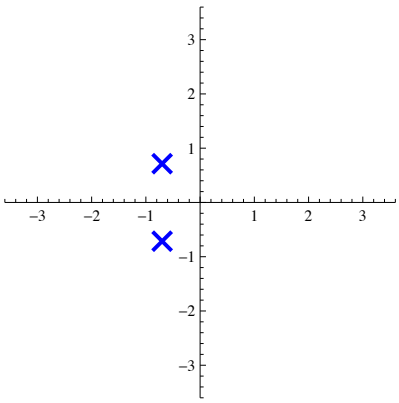
Andra filtertyper

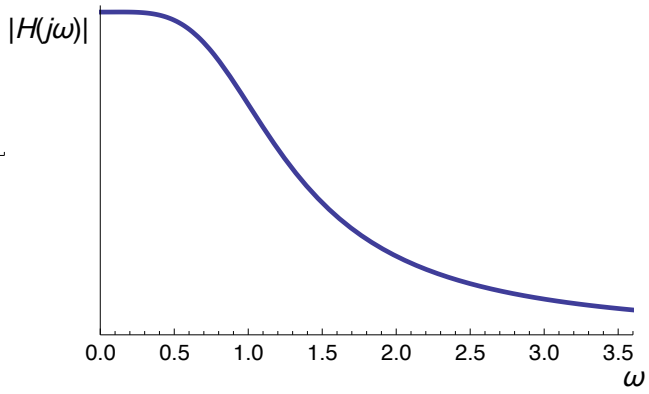


Mathematica demo

Lågpass (LP)

Exempel: $H(s) = \dfrac{1}{1+1.4s+s^2}$

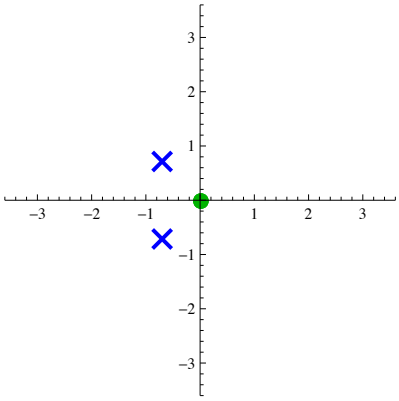


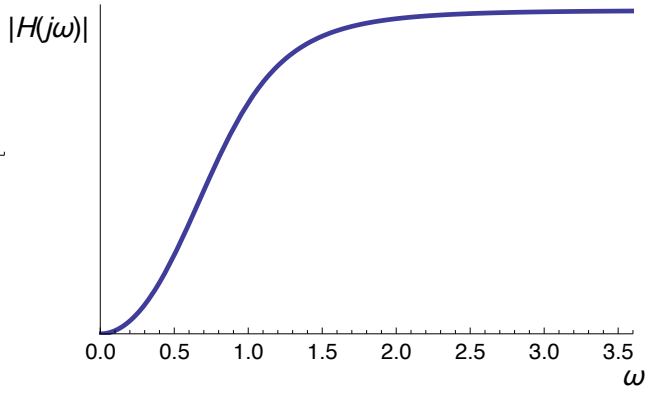


poler

Högpass (HP)

Exempel: $H(s) = \dfrac{s^2}{1+1.4s+s^2}$

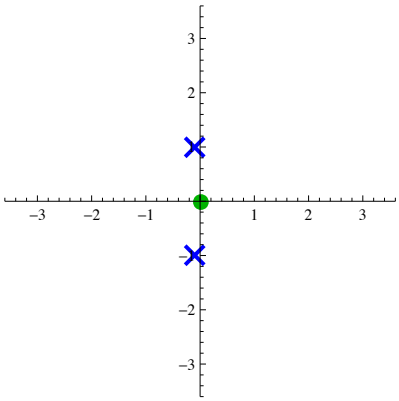


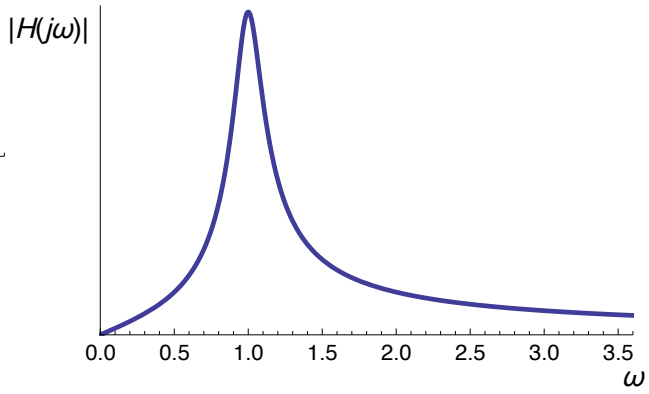


poler
nollställen

Bandpass (BP)

Exempel: $H(s) = \dfrac{s}{1 + 0.2s + s^2}$

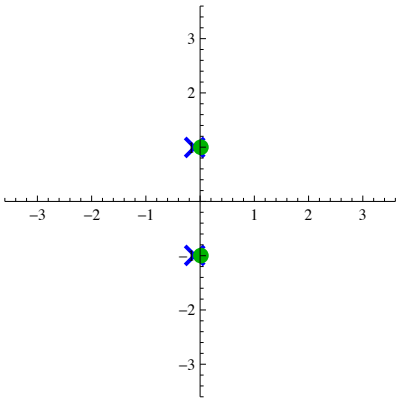


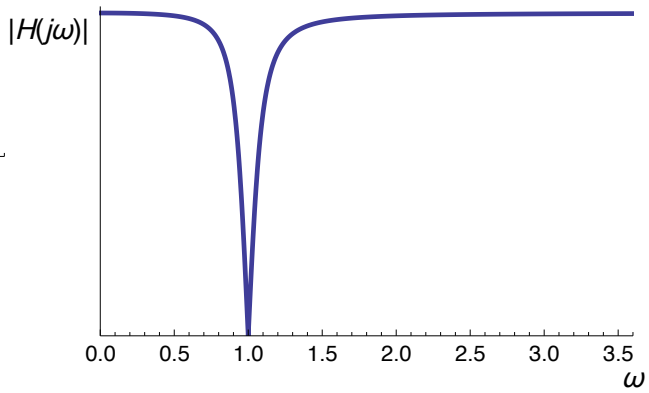


poler
nollställen

Bandspärr (BS)

Exempel: $H(s) = \dfrac{1 + s^2}{1 + 0.2s + s^2}$





poler
nollställen

Filtertabeller

De vanligaste typerna av filter kan konstrueras från tabeller med filterpolynom

Butterworth-polynom:

Ordning	Polynom $P(a)$
1	$1+a$
2	$1+1.414a+a^2$
3	$1+2a+2a^2+a^3$ $=(1+a)\left(1+a+a^2\right)$
4	$1+2.613a+3.414a^2+2.613a^3+a^4$ $=\left(1+0.765a+a^2\right)\left(1+1.848a+a^2\right)$

Chebyshev-polynom av typ 1 med 3 dB rippel:

Ordning	Polynom $P(a)$
2	$1.414+1.287a+2a^2$ $=1.414\left(1+0.910a+1.414a^2\right)$
3	$1+3.711a+2.384a^2+4a^3$ $=(1+3.355a)\left(1+0.355a+1.192a^2\right)$
4	$1.414+3.231a+9.348a^2+4.644a^3+8a^4$ $=1.414\left(1+0.188a+1.108a^2\right)\left(1+2.096a+5.108a^2\right)$

Filterkonstruktion med hjälp av tabell

LP med gränsfrekvens $\omega_{ö}$:

$$H_{LP}(s) = \frac{1}{P\left(s/\omega_{ö}\right)}$$

HP med gränsfrekvens ω_u:

$$H_{HP}(s) = \frac{1}{P\left(\omega_u/s\right)}$$

Exempel: Konstruera ett andra ordningens Butterworth HP-filter med gränsfrekvens 10 rad/s □

Resultat: som bild 16 med ändrad ω-skala

Läs & lös

Läs:

- Bengtsson s 359–373 (linjära system, poler och nollställen)
- Bengtsson s 410–414, 423–425, 430 (filterteori)

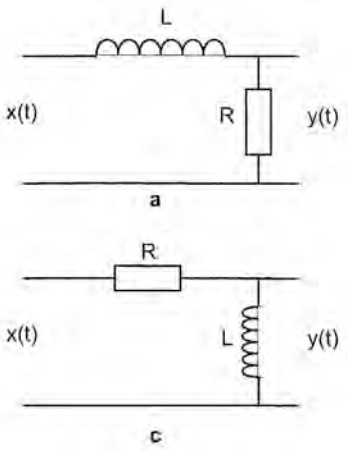
Lös:

- Bengtsson uppg 10.7
- Bengtsson uppg 12.1 och 12.17a–b

10.7 Vilka av följande system är stabila:

a $H(s)=\dfrac{s^2+4}{s^2+2s+9}$ b $H(s)=\dfrac{s}{s^2-6s+8}$ c $H(s)=\dfrac{3s-7}{s^2+3s-9}$

12.1 Ange överföringsfunktionen för filtren nedan. Ange om det är låg- eller högpassfilter samt ω_0.



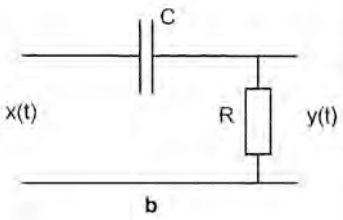


Fig 12.27 Första ordningens passiva filter

12.17a I tabell 12.2 är Chebyshev-filtret av ordning 3 skrivet som ett tredjegradspolynom. Separera detta polynom så att du kan skriva filtret som en seriekoppling av ett första och ett andra ordningens filter.

12.17b Använd transformeringsreglerna i avsnitt 12.5 för att transformera filtret till ett lågpassfilter med gränsfrekvensen 1 kHz.

12.17c Realisera detta filter med hjälp av en första och en andra ordningens Sallen-Key-länk.

$$\frac{0{,}2506}{s^3 + 0{,}5972s^2 + 0{,}9283s + 0{,}2506}$$

Föreläsning 13
Implementering av analoga filter

Erik Agrell 2017-10-04

Innehåll:

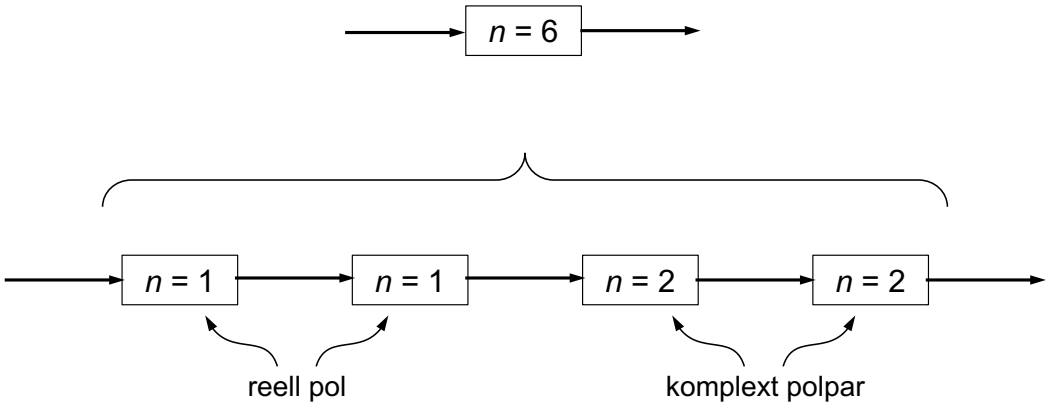
1. Kaskadkopplade filterlänkar
2. Sallen-Key-kopplingen

Bilder av Erik Agrell

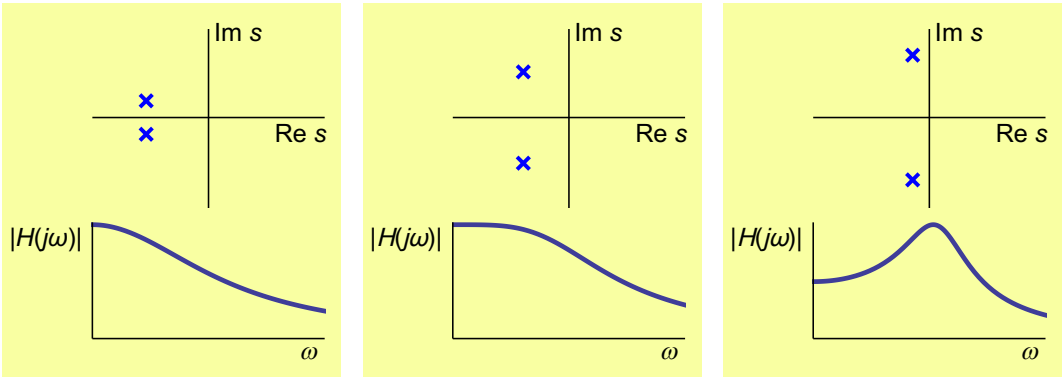
1. Kaskadkopplade filterlänkar

Alla filter kan implementeras genom att kaskadkoppla 1a och 2a ordningens filterlänkar.

Exempel 1:

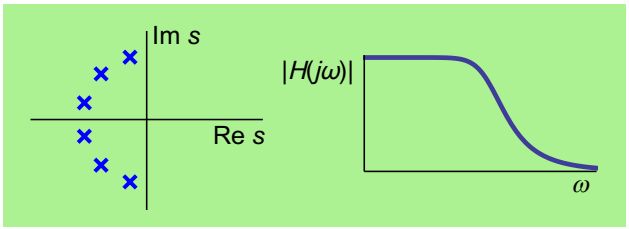


Exempel 2



Tre olika länkar av ordning 2…

… ger tillsammans ett bra filter av ordning 6:

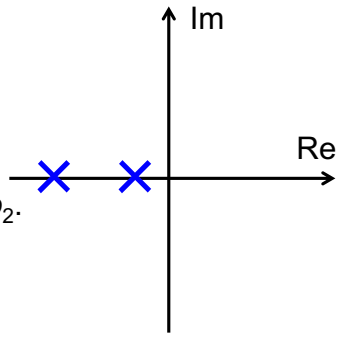


Exempel 3

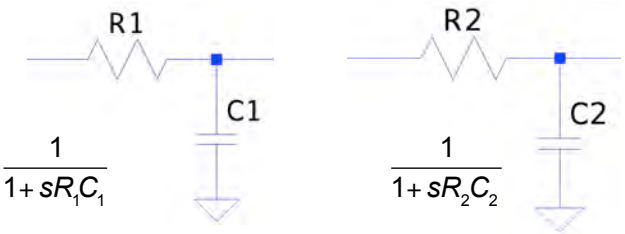
Vi vill implementera

$$H(s) = \frac{1}{(1 - s/p_1)(1 - s/p_2)}$$

med två reella, negativa poler p_1 och p_2.

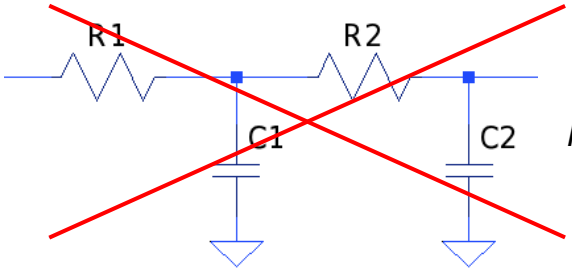


1. Skapa två LP-länkar enligt nedan, där $R_1C_1 = -1/p_1$ och $R_2C_2 = -1/p_2$



2. Kombinera länkarna…

Försök 1:

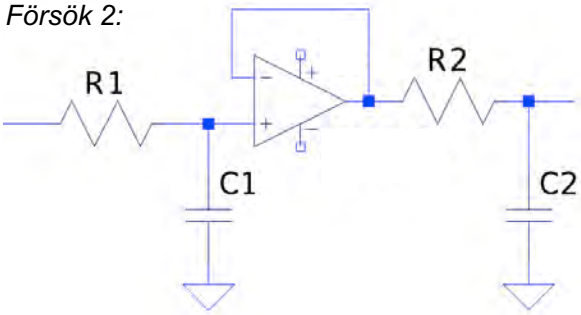


$$H(s) = \frac{1}{(1+sR_1C_1)(1+sR_2C_2)+sR_1C_2}$$

??

Länk 2 påverkar länk 1!

Försök 2:



$$H(s) = \frac{1}{(1+sR_1C_1)(1+sR_2C_2)}$$

OK!

Komponentvärden

Tumregler:

- Motstånd väljs ofta ur E12-serien i intervallet ~100 Ω – 100 kΩ
- Kondensatorer väljs ofta ur E6-serien i intervallet ~100 pF – 10 µF

(undantag finns)

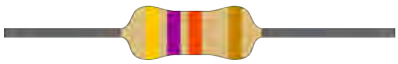
⇒ Med *R* och *C* i dessa intervall kan man implementera tidskonstanter *RC* i intervallet ~10^{-8} – 1 s (frekvenser ~1 – 10^{8} rad/s)

E12	E6
10	10
12	
15	15
18	
22	22
27	
33	33
39	
47	47
56	
68	68
82	

Det är bättre att välja *C* och beräkna *R* än tvärt om, eftersom tillgängliga *R*-värden ligger tätare (typiskt E6-serien)

Avrundningsfel

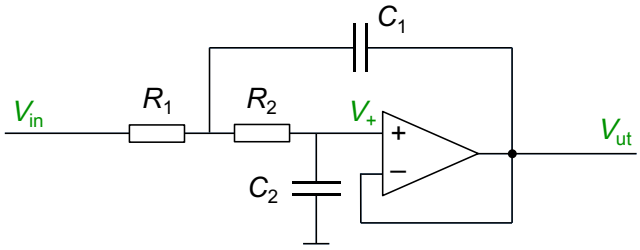
Det blir alltid avrundningsfel när man väljer komponentvärden.
Vad kan man göra åt det?



Det finns åtminstone tre möjliga strategier:

1. Använd parallell- eller seriekoppling för att skapa värden mellan E12-nivåerna
2. Prova andra värden på C tills avrundningsfelen i R blir små
3. Simulera och kolla om avrundningsfelen spelar någon roll

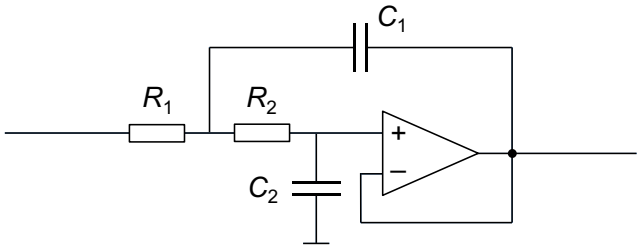
2. Sallen-Key-filtret



Vilken typ av filter? HP, LP eller något annat?

- Spänningsföljare ⇒ $V_{ut} = V_+$
- Låga frekvenser
  ⇒ kondensatorer spärrar
  ⇒ $V_+ = V_{in}$
  ⇒ $H(0) = 1$
- Höga frekvenser
  ⇒ kondensatorer leder
  ⇒ $V_+ = 0$
  ⇒ $H(\infty) = 0$

Svar: Lågpass!



Vilken överföringsfunktion?

Svar: $$H(s) = \frac{1}{1 + s\left(R_1 + R_2\right)C_2 + s^2 R_1 R_2 C_1 C_2}$$

Dimensionering av komponentvärden

Med Sallen-Key-kopplingen implementeras

- två reella poler *eller*
- ett komplext polpar

Utimpedans 0 $\Rightarrow$ kan kaskadkopplas

Hur väljer man komponenter?

1. Skriv den önskade överföringsfunktionen som

$$H(s) = \frac{1}{1 + \alpha s + \beta s^2}$$

2. Välj komponenter så att

$$\alpha = \left(R_1 + R_2\right)C_2$$

$$\beta = R_1 R_2 C_1 C_2$$

Exempel

Konstruera ett tredje ordningens LP-filter av Butterworth-typ med gränsfrekvens 2 kHz genom att kombinera ett Sallen-Key-filter av ordning 2 och ett passivt första ordningens LP-filter. Motstånd skall tas ur E12-serien och kondensatorer ur E6-serien.

Vi behöver en tabell över Butterworth-polynom, från kursens formelsamling:

Ordning	Polynom $P(a)$
1	$1+a$
2	$1+1.414a+a^2$
3	$1+2a+2a^2+a^3$ $=(1+a)\left(1+a+a^2\right)$
4	$1+2.613a+3.414a^2+2.613a^3+a^4$ $=\left(1+0.765a+a^2\right)\left(1+1.848a+a^2\right)$

Spice demo

Veckosammanfattning

Efter kursvecka 6 behärskar ni…

- Analoga filter
  - Poler och nollställen
  - Överföringsfunktion och frekvensfuktion
  - Frekvenstransformering
  - Butterworthfilter
  - LP-, HP-, BP- och BS-filter
  - Filterkonstruktion med hjälp av tabell
- Implementation av filter
  - Filterkonstruktion med kaskadkoppling
  - Sallen-Key-länken
  - Val av komponentvärden

Läs & lös

Läs:

- Bengtsson s 419–420 (Sallen-Key-kopplingen)
- Lab-PM, lab 6

Lös:

- Bengtsson uppg 12.17c
- Förberedelseuppgifter lab 6

12.17a I tabell 12.2 är Chebyshev-filtret av ordning 3 skrivet som ett tredjegradspolynom. Separera detta polynom så att du kan skriva filtret som en seriekoppling av ett första och ett andra ordningens filter.

12.17b Använd transformeringsreglerna i avsnitt 12.5 för att transformera filtret till ett lågpassfilter med gränsfrekvensen 1 kHz.

12.17c Realisera detta filter med hjälp av en första och en andra ordningens Sallen-Key-länk.

$$\frac{0{,}2506}{s^3 + 0{,}5972s^2 + 0{,}9283s + 0{,}2506}$$

Föreläsning 14
Förstärkardimensionering och stabilitet

Erik Agrell 2017-10-10

Innehåll:

1. Icke ideal operationsförstärkare
2. Kaskadkoppling
3. Stabilitet
4. Gästföreläsning

Bilder av Bengt Molin och Erik Agrell

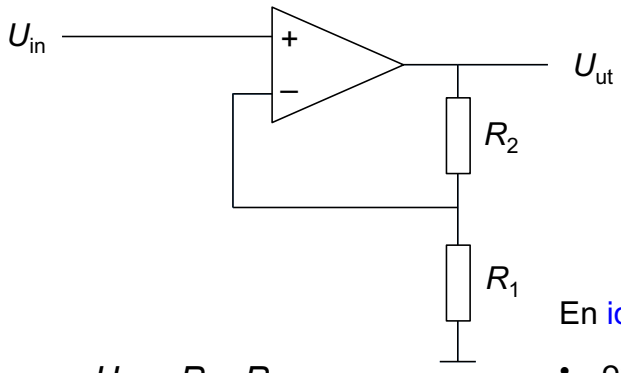
Veckosammanfattning

Efter kursvecka 6 behärskar ni…

- Analoga filter
  - Poler och nollställen
  - Överföringsfunktion och frekvensfuktion
  - Frekvenstransformering
  - Butterworthfilter
  - LP-, HP-, BP- och BS-filter
  - Filterkonstruktion med hjälp av tabell
- Implementation av filter
  - Filterkonstruktion med kaskadkoppling
  - Sallen-Key-länken
  - Val av komponentvärden

1. Icke ideal operationsförstärkare

Icke inverterande förstärkare:



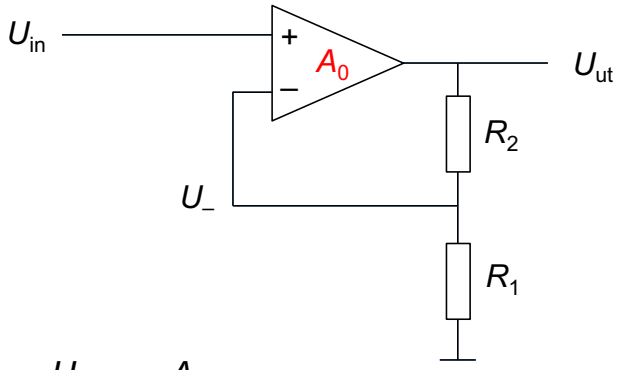
$$A = \frac{U_{ut}}{U_{in}} = \frac{R_1 + R_2}{R_1}$$

… om OPn är ideal

En ideal OP har:

- oändlig råförstärkning
- oändlig bandbredd
- oändlig inimpedans
- utimpedans 0

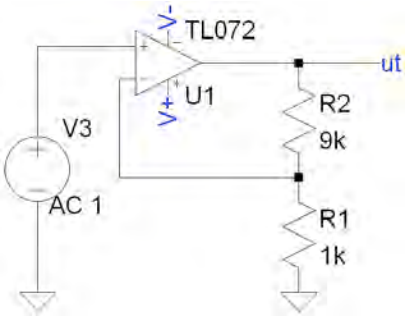
En verklig op har ändlig råförstärkning



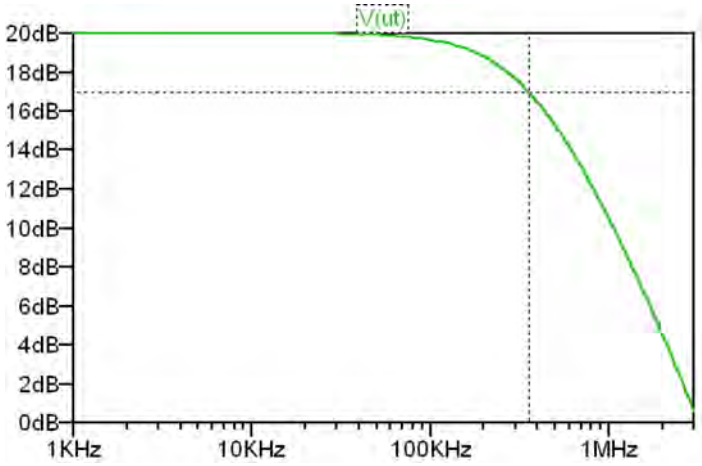
$$A = \frac{U_{ut}}{U_{in}} = \frac{A_0}{1 + \beta A_0}$$ där motkopplingsfaktorn $$\beta = \frac{R_1}{R_1 + R_2}$$

Gränsvärde (ideal OP): Om $A_0 \to \infty$, så är $$A = \frac{1}{\beta} = \frac{R_1 + R_2}{R_1}$$ OK!

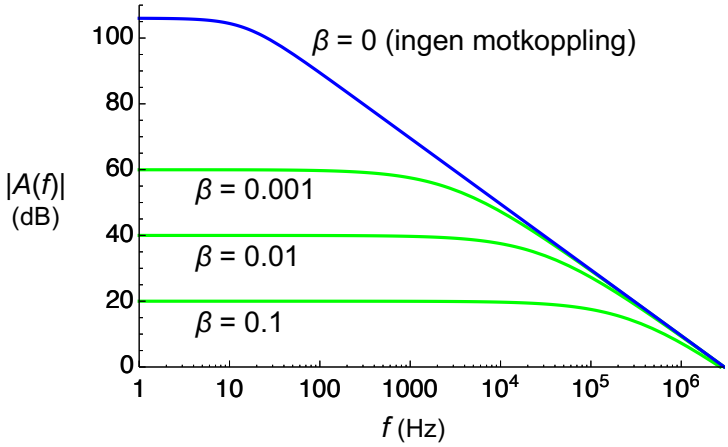
En verklig op har ändlig bandbredd också



Spice demo



Förstärkningens frekvensberoende

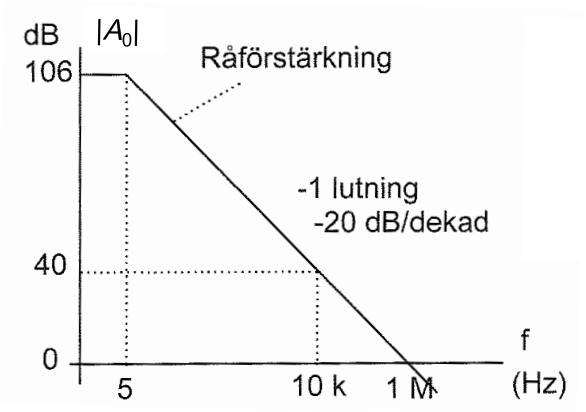


En verklig OP har:

- ändlig råförstärkning A_{max} (typiskt 10^5–10^6 gånger vid DC)
- låg bandbredd $f_ö$ (typiskt 1–20 Hz!!)

Råförstärkningen
(utan motkoppling)

[Molin s 72]



logaritmisk frekvensskala

- En OP har avsiktligt mycket låg gränsfrekvens
- Metoden kallas intern frekvenskompensering och ger
  - god stabilitet vid motkoppling
  - men låg bandbredd utan motkoppling

Låga och höga frekvenser

Från bild 3:

$$A(f) = \frac{A_0(f)}{1+\beta A_0(f)}$$

- Låga frekvenser:

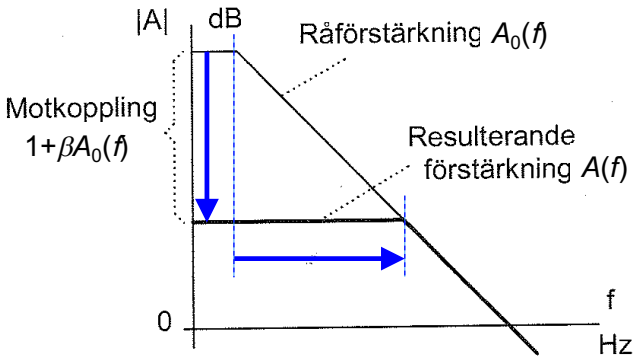
$$A(f) \approx \frac{1}{\beta}$$

  - oberoende av f
  - oberoende av A_0
  - okänslig för ickelinjär råförstärkning

- Höga frekvenser:

$$A(f) \approx A_0(f)$$

  - oberoende av β



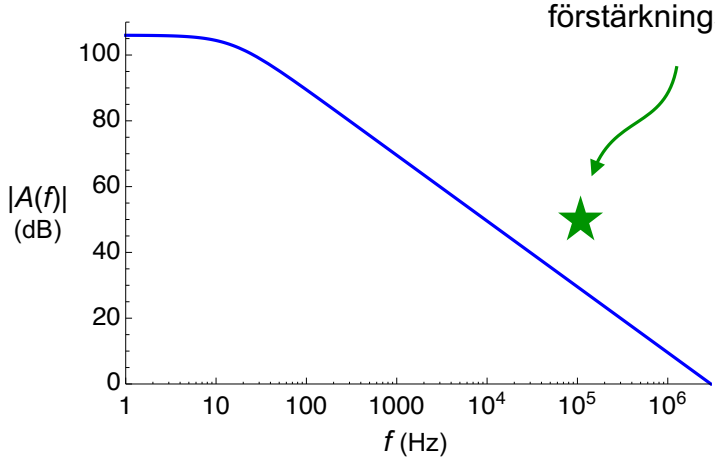
[Molin s 131]

- Motkoppling minskar förstärkningen
- Motkoppling ökar bandbredden

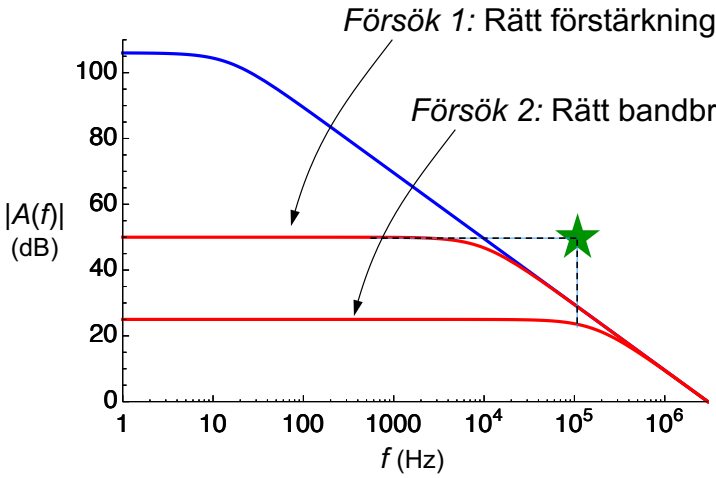
Produkten förstärkning × bandbredd är konstant

Dagens gåta

Hur konstruerar man förstärkare med prestanda utanför förstärknings-bandbredds-linjen?

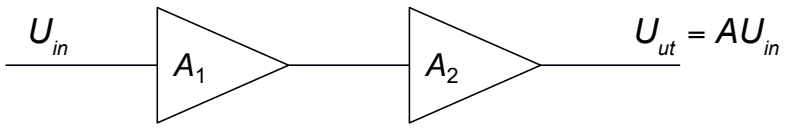


Två misslyckade försök



…bättre svar om en stund…

2. Kaskadkoppling



$$A = A_1 \cdot A_2$$

eller om förstärkningarna är frekvensberoende

$$A(f) = A_1(f) \cdot A_2(f)$$

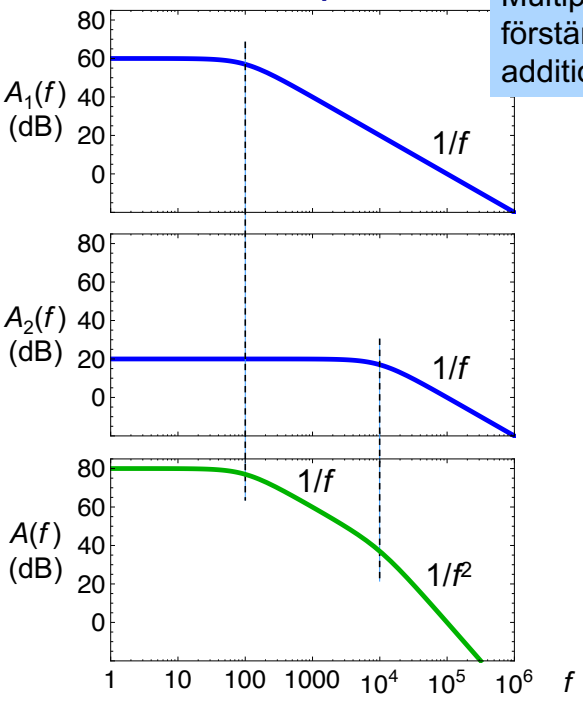
$$\Rightarrow \underbrace{20\log A(f)}_{\text{dB}} = 20\log A_1(f)A_2(f)$$

$$= \underbrace{20\log A_1(f)}_{\text{dB}} + \underbrace{20\log A_2(f)}_{\text{dB}}$$

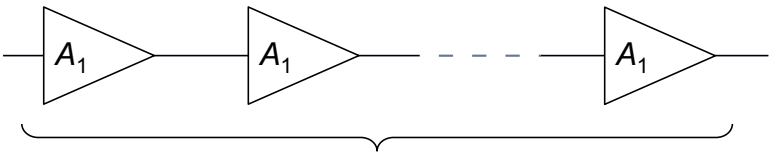
Multiplikation av förstärkning motsvarar addition i dB-skala

Exempel

Multiplikation av förstärkning motsvarar addition i dB-skala

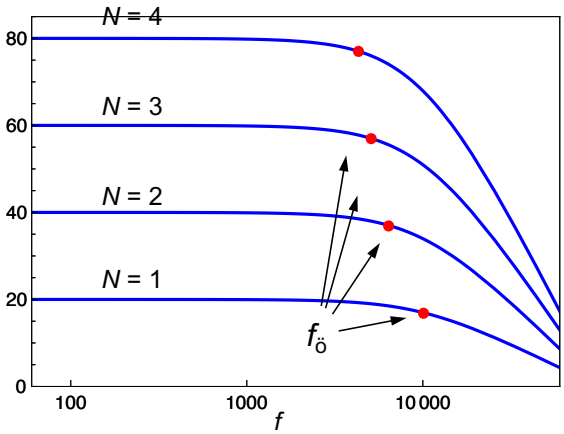


Kaskadkopplade lika steg



$$A(f) = A_1(f)^N$$

$$\Rightarrow \underbrace{20\log A(f)}_{\text{dB}} = N \cdot \underbrace{20\log A_1(f)}_{\text{dB}}$$



När flera steg kaskadkopplas, så…

- ökar förstärkningen (mycket), $A = A_1^N$
- minskar bandbredden (måttligt)

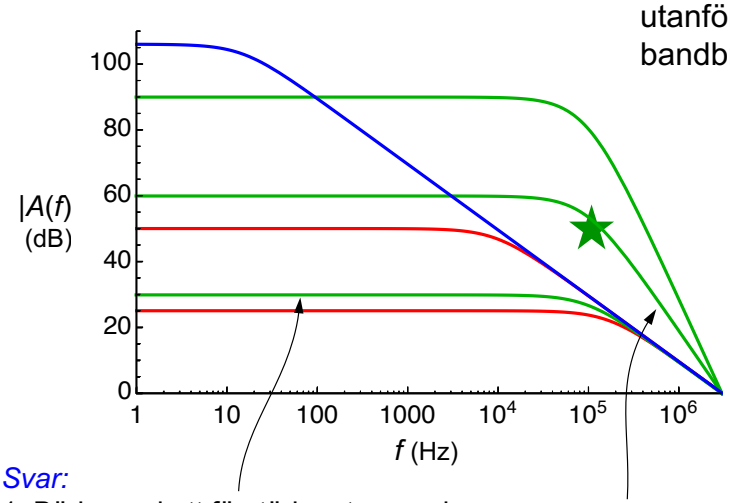
Specialfall: När första ordningens steg kaskadkopplas, så minskar bandbredden som

$$f_{ö} = f_{ö1}\sqrt{2^{1/N} - 1}$$

Produkten förstärkning × bandbredd är inte konstant för kaskadkopplade steg

Svar på gåtan (bild 9)

Fråga: Hur konstruerar man förstärkare med prestanda utanför förstärknings-bandbredds-linjen?



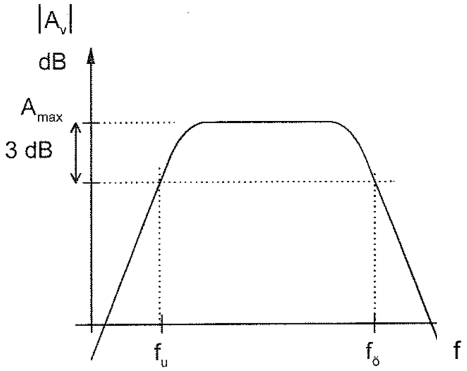
Svar:

1. Börja med ett förstärkarsteg med högre bandbredd än den önskade

2. Kaskadkoppla flera sådana steg tills förstärkningen är tillräcklig

Kaskadkoppling med filtrering

En förstärkare bör förstärka signaler inom ett önskat frekvensintervall och dämpa signaler utanför



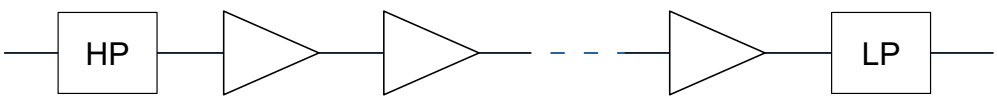
⇒ Praktiska förstärkare kombinerar ofta:

- kaskadkoppling (ger tillräcklig förstärkning)
- HP-länk (ger f_u)
- LP-länk (ger $f_ö$)

I vilken ordning placeras stegen?

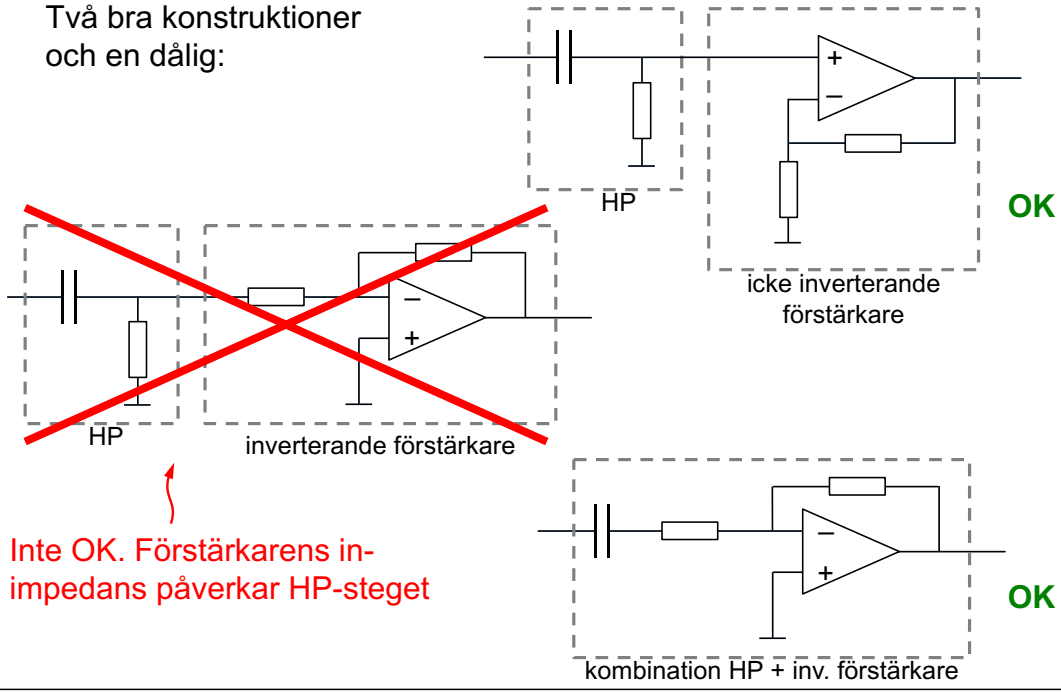
Man brukar lägga HP-steget först, för att slippa DC-nivåer.

Man brukar lägga LP-steget sist, för att reducera eventuella störningar från förstärkarkedjan.



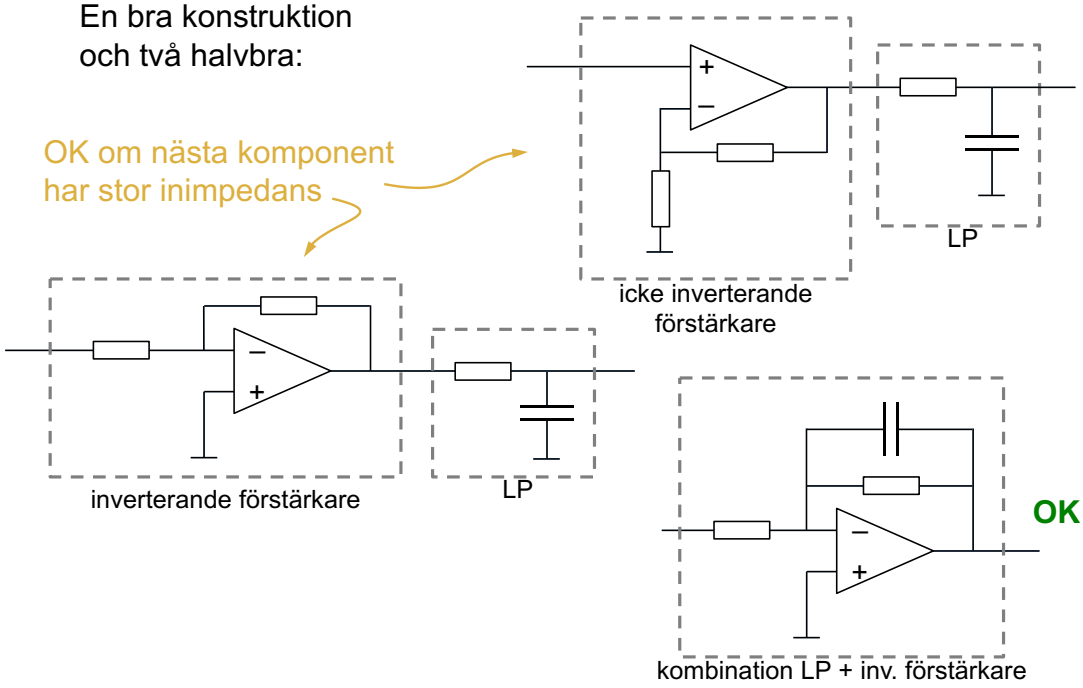
HP-steget

Två bra konstruktioner och en dålig:



LP-steget

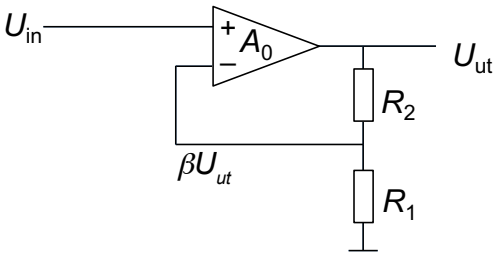
En bra konstruktion och två halvbra:



3. Stabilitet

Ett motkopplat system är instabilt om $|\beta A_0(f)| > 1$ (d.v.s. 0 dB) vid den frekvens där $\arg\beta A_0(f) = -180^\circ$

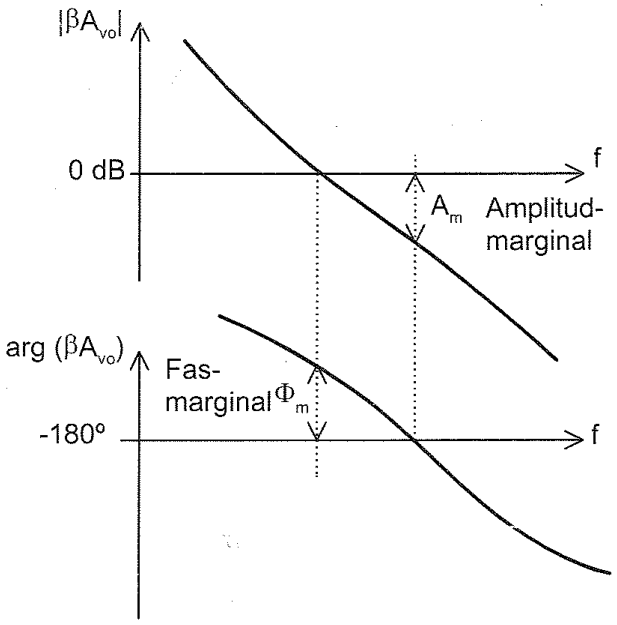
Exempel:



Motkopplingsfaktor $\beta = \dfrac{R_1}{R_1 + R_2}$

- Amplitudmarginal är skillnaden mellan $|\beta A_0(f)|$ i dB och 0 dB vid den frekvens där $\arg\beta A_0(f) = -180^\circ$
- Fasmarginal är skillnaden mellan $\arg\beta A_0(f)$ och -180° vid den frekvens där $|\beta A_0(f)| = 0$ dB

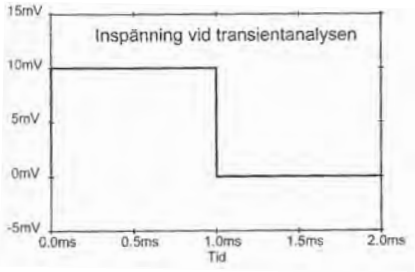
Amplitud- och fasmarginal



[Molin s 143]

Exempel: utsignaler vid olika motkoppling

Förstärkarens insignal:

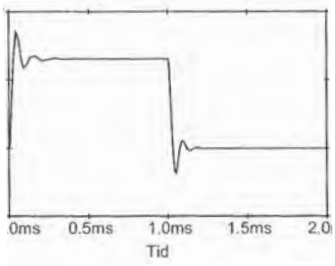


Tumregler: Sikta på

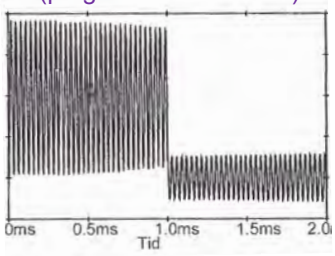
- minst 6 dB amplitudmarginal
- minst 45° fasmarginal

Utsignaler vid olika motkoppling:

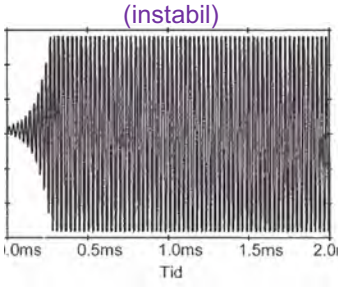
Fasmarginal 45°



Fasmarginal 0°
(på gränsen till instabil)



Fasmarginal <0°
(instabil)



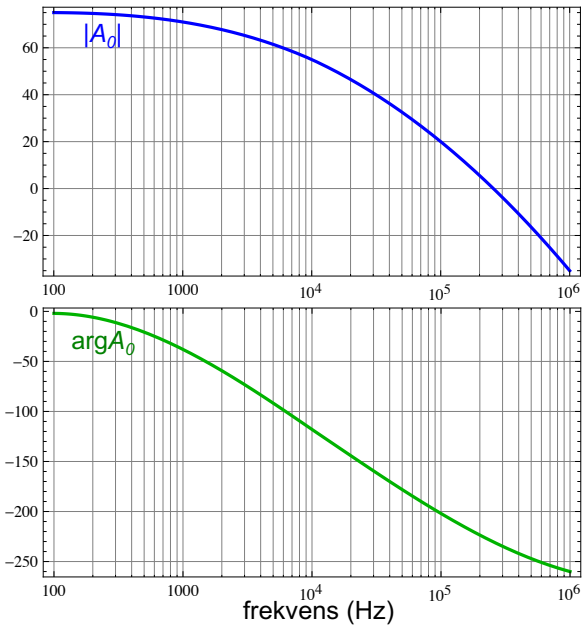
Exempel

Kurvorna visar råförstärkningen (belopp och fas) för en förstärkare.

(a) Vid vilket värde på motkopplingsfaktorn β kommer förstärkaren att självsvänga?

(b) Bestäm amplitud- och fasmarginal om förstärkaren motkopplas med ett resistivt nät som ger spänningsdelning 1/100

(c) Vilket är det högsta värdet på β som kan användas om fasmarginalen skall vara åtminstone 45°?



[Molin s 146, modifierad]

Svar: (a) $\beta > -32$ dB, (b) 8 dB resp. 20°, (c) $\beta = 50$ dB

4. Gästföreläsning

Har det vi lär oss i den här kursen något med verkligheten att göra?

Svar i nästa vecka…

Radiolänk: Design och arbetsflöde
av Henrik Boklund och Magnus Wisell

Gästföreläsning i Elektriska system
tisdag 17 okt kl 8:15 i J243

ERICSSON

Gästföreläsarna är radiodesigner på Ericsson Lindholmen. De kommer att berätta om hur ingenjörer jobbar på Ericsson för att utveckla nya system för trådlös kommunikation. Föredraget behandlar principerna för radiokommunikationen i moderna mobiltelefonisystem. Utvecklingsprocessen beskrivs också, från kravspecifikation via design och simulering till felsökning och verifiering.

Läs & lös

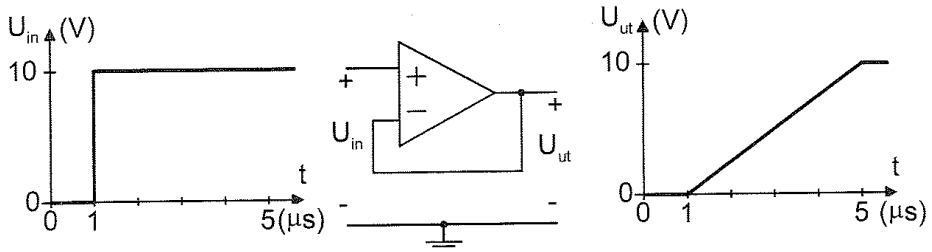
Läs:

- Molin s 71–75 (icke ideal op)
- Molin s 130–137 (motkoppling)
- Molin s 141–147 (stabilitet)

Lös:

- Uppgift (c) på bild 24
- Molin uppgift 3.5, 3.6, 6.4
- Inlämningsuppgift 4

3.5 En spänningsföljare matas med en stegspänning. Utspänning enligt figuren nedan erhålls. Hur stor är förstärkarens slew rate?



3.6 En operationsförstärkare 741 som är internt kompenserad har följande data:

Råförstärkning	200 000 ggr
Övre gränsfrekvens	5 Hz
CMRR	90 dB
Slew rate	0,5 V/µs

Man önskar konstruera en förstärkare med 100 gångers förstärkning som skall kunna förstärka en sinusspänning med 100 mV toppvärde på förstärkarens ingång.

a Vilken högsta frekvens kan användas utan att utspänningen förvrängs på grund av slewrate?

b Vilken övre gränsfrekvens får förstärkaren om operationsförstärkarens frekvenskarakteristik beaktas?

c Är det slew rate eller operationsförstärkarens frekvenskurva som begränsar maximalt användbar frekvens i detta fall?

6.4 En förstärkare med råförstärkningen

$$A_{vo} = 1000 \cdot \frac{1}{1+j\frac{f}{100}} \cdot \frac{1}{1+j\frac{f}{1000}}$$

spänning-spänning-motkopplas.
Vilken är den största resistiva motkoppling β och största resulterande förstärkning vid låga frekvenser som kan användas om fasmarginal större än 45 grader önskas.

a Uppskatta β_{max} approximativt med Bodediagram.

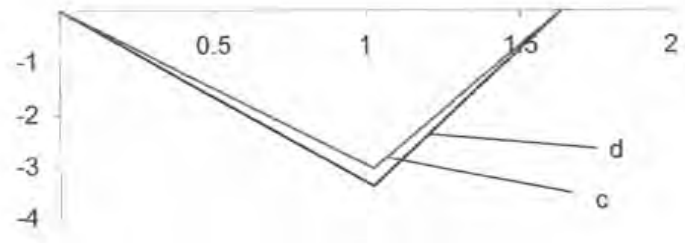
b Beräkna β_{max} exakt genom att beräkna fas och belopp för uttrycket A_{vo}.

Svar på vissa uppgifter

som rekommenderats på föreläsningar

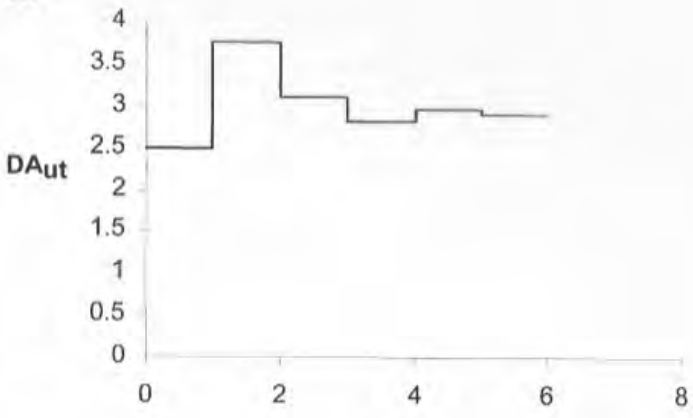
Bengtsson:

4.10a 0- 5 V **4.10b** 1,6384 ms **4.10c&d** se figur nedan (notera att resultatet blir detsamma i båda fallen).



4.11a Sex jämförelser 4.11c 100100_2 respektive 0,0275 V

4.11b



4.13c $\Delta_1 = 2{,}5$ V, $\Delta_2 = 1{,}25$ V, $\Delta_3 = 0{,}625$ V, $\Delta_4 = 0{,}3125$ V

4.16a 15 stycken b 0,3125 V, fyra bitar c 1110_2 d 3,594 – 3,9062 V

Molin:

3.5: 2.5 µs

3.6: (a) 8.0 kHz. (b) 10 kHz. (c) Slew rate begränsar.

6.4: (a) $\beta_{max} \approx 0.01$, förstärkning 100 (20 dB). (b) $\beta_{max} = 0.018$, förstärkning 18 (25 dB).

8.2: (a) 1.4 V. (b) 0.10 mA.

9.2 (om λ=0): (a) 15 kΩ. (b) ±150 mV.

Föreläsning 15
Kursinformation och tentamensräkning

Erik Agrell 2017-10-11

Innehåll:

1. Kursinformation
2. Uppgifter ur tidigare tentamen

1. Kursinformation

- Fredag 13 okt 16:00: deadline för inlämningsuppgift 4
- Tisdag 17 okt 8:15: F16, gästföreläsning (se nästa bild, rekommenderas!)
- Tisdag 17 okt 13:15–17:00: extra labtillfälle (frivilligt)
- Fredag 20 okt 16:00: deadline för labrapport (för sen labrapport ger max 2 p)
- Torsdag 26 okt 14:00: tentamen
- Måndag 30 okt?: kursenkät

Gästföreläsning

Har det vi lär oss i den här kursen något med verkligheten att göra?

Svar i nästa vecka…

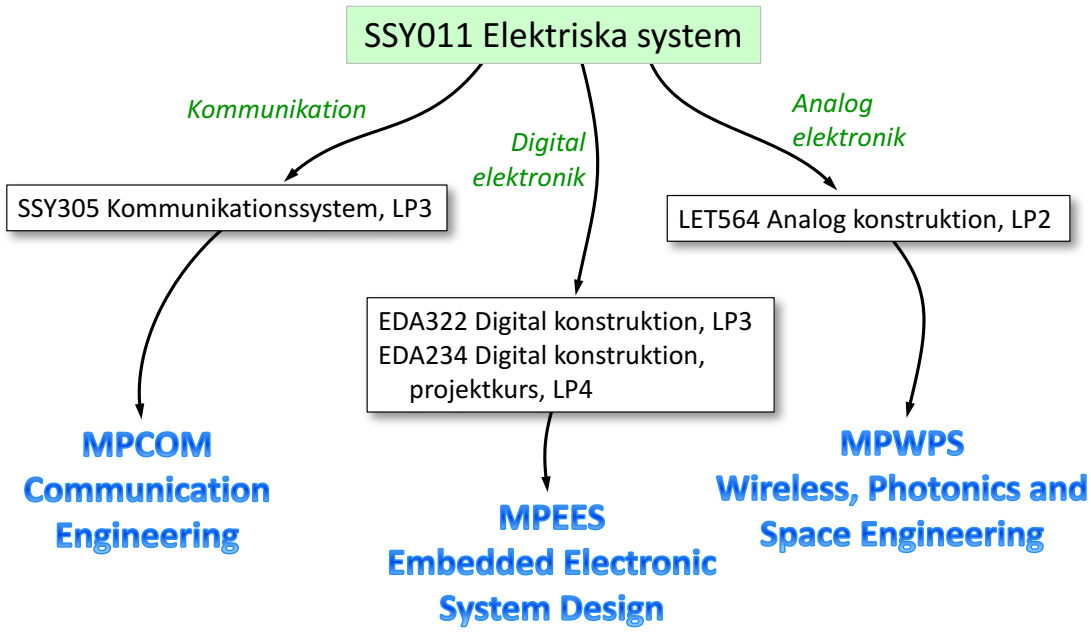
Radiolänk: Design och arbetsflöde
av Henrik Boklund och Magnus Wisell

Gästföreläsning i Elektriska system
tisdag 17 okt kl 8:15 i J243

ERICSSON

Gästföreläsarna är radiodesigner på Ericsson Lindholmen. De kommer att berätta om hur ingenjörer jobbar på Ericsson för att utveckla nya system för trådlös kommunikation. Föredraget behandlar principerna för radiokommunikationen i moderna mobiltelefonisystem. Utvecklingsprocessen beskrivs också, från kravspecifikation via design och simulering till felsökning och verifiering.

Läsa vidare efter kursen?



Examination och betygsättning
(från F1)

- Tentamen ger max 50 poäng.
- Laborationen (utförandet, förberedelseuppgifter och rapporten), bedöms med max 4 poäng. Minst 2 poäng krävs för godkänd labkurs.
- Inlämningsuppgifter, som inlämnas i tid, ger max 1 poäng per uppgift
- Bonuspoäng från lab och inlämningsuppgifter får tillgodoräknas vid ordinarie tentamen (men inte senare).
- För godkänd kurs krävs minst 20 poäng på tentamen (inkl ev bonus) och minst 2 poäng på laborationen.
- För betyg 3, 4 och 5 krävs 20, 30 resp. 40 poäng.

Vad kommer på tentan?

- Kursinnehållet definieras av föreläsningar, laboration, inlämningsuppgifter och litteraturen.
- Förståelse är viktigare än utantillkunskap.
- Tidigare tentor finns på pingpong men är inte 100% relevanta. Viss kursutveckling görs varje år.
- En formelsamling kommer att bifogas tentan.

Formelsamling

Decibel, definition

$$\mathrm{dB} = 10\log_{10}\frac{P_1}{P_0}$$

Decibel, speciella referensnivåer

$$\mathrm{dBW} = 10\log_{10}\frac{P}{1\mathrm{W}}$$

$$\mathrm{dBm} = 10\log_{10}\frac{P}{1\mathrm{mW}}$$

$$\mathrm{dBV} = 20\log_{10}\frac{U_{\mathrm{eff}}}{1\mathrm{V}}$$

$$\mathrm{dB\ SPL} = 20\log_{10}\frac{p}{20\mu\mathrm{Pa}}$$

(94 dB SPL $\leftrightarrow$ 1 Pa)

FET, n-kanal

$$I_{\mathrm{D}} = \begin{cases} 0 & \text{if } U_{\mathrm{GS}} \le U_{\mathrm{T}} \\ \beta(U_{\mathrm{GS}} - U_{\mathrm{T}})^2 & \text{if } 0 \le U_{\mathrm{GS}} - U_{\mathrm{T}} \le U_{\mathrm{DS}} \end{cases}$$

FET, p-kanal

$$I_{\mathrm{D}} = \begin{cases} 0 & \text{if } -U_{\mathrm{GS}} \le -U_{\mathrm{T}} \\ -\beta(-U_{\mathrm{GS}} + U_{\mathrm{T}})^2 & \text{if } 0 \le U_{\mathrm{GS}} - U_{\mathrm{T}} \le U_{\mathrm{DS}} \end{cases}$$

Kaskadkopplade 1:a ordningens förstärkarsteg

$$A = A_1^N$$

$$f_{\ddot{o}} = f_{\ddot{o}1}\sqrt{2^{1/N} - 1}$$

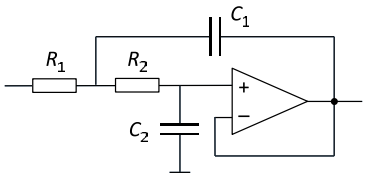
Filterkonstruktion, tabellmetoden

$$\text{LP:} \quad H(s) = \frac{1}{P(s/\omega_{\ddot{o}})}$$

$$\text{HP:} \quad H(s) = \frac{1}{P(\omega_{\mathrm{u}}/s)}$$

Sallen-Key-filter

$$H(s) = \frac{1}{1 + s(R_1 + R_2)C_2 + s^2 R_1 R_2 C_1 C_2}$$



Butterworth-filter

Ordning	Polynom $P(a)$
1	$1 + a$
2	$1 + 1.414a + a^2$
3	$1 + 2a + 2a^2 + a^3$ $= (1 + a)(1 + a + a^2)$
4	$1 + 2.613a + 3.414a^2 + 2.613a^3 + a^4$ $= (1 + 0.765a + a^2)(1 + 1.848a + a^2)$

Chebyshev-filter typ 1 med 3 dB rippel

Ordning	Polynom $P(a)$
2	$1.414 + 1.287a + 2a^2$ $= 1.414(1 + 0.910a + 1.414a^2)$
3	$1 + 3.711a + 2.384a^2 + 4a^3$ $= (1 + 3.355a)(1 + 0.355a + 1.192a^2)$
4	$1.414 + 3.231a + 9.348a^2 + 4.644a^3 + 8a^4$ $= 1.414(1 + 0.188a + 1.108a^2)(1 + 2.096a + 5.108a^2)$

Komponentvärden

E12	E6
10	10
12	
15	15
18	
22	22
27	
33	33
39	
47	47
56	
68	68
82	

Några råd inför tentan

- Förklara alla led
- Definiera axlarna på diagram
- Kolla att enheterna stämmer
- Gör en rimlighetsbedömning
- Skriv läsligt och begripligt

Poängbedömning

- OK uträkningar, rimligt svar ⇒ höga poäng
  – även om svaret är fel!
- Allvarliga fel i uträkningarna ⇒ låga poäng
  – även om svaret är rätt!
- Orimligt svar ⇒ låga poäng
  – en kommentar om orimligheten kan hjälpa

Kursenkät

Kursenkätens syfte:

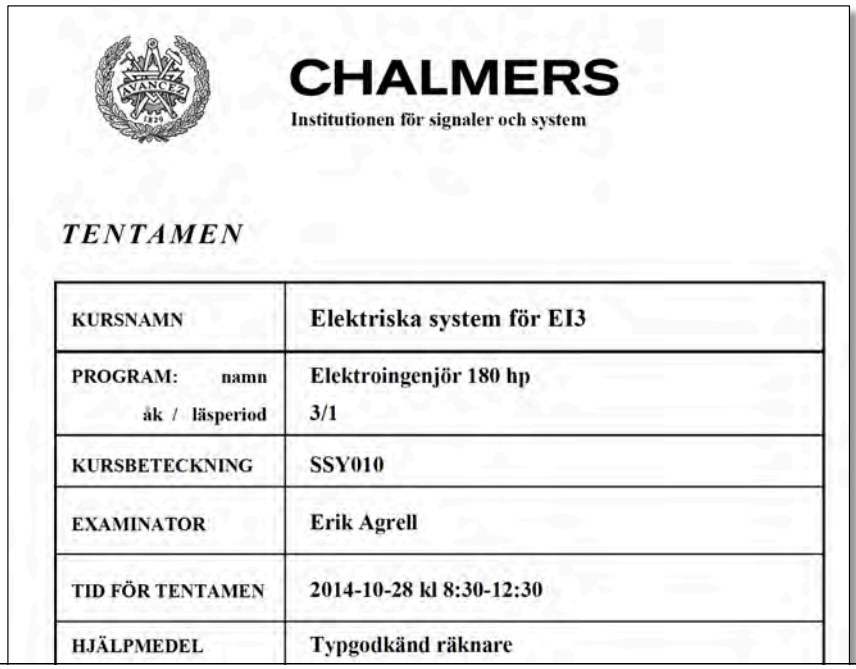
- Hjälpa läraren med kursutveckling (hur kan olika kursmoment förbättras?)
- Hjälpa Chalmers med kursutbudet (behålla eller lägga ned kursen? behålla eller byta ut lärare?)

Veckosammanfattning

Efter kursvecka 7 behärskar ni…

- Icke ideal operationsförstärkare
  - Ändlig råförstärkning
  - Ändlig bandbredd
  - Frekvensberoende vid motkoppling
  - Förstärknings-bandbredds-produkt
- Förstärkarkonstruktion
  - Kaskadkoppling
  - Kombination av förstärkarsteg och filter
- Stabilitet vid motkoppling
  - Stabilitetskriteriet
  - Amplitudmarginal
  - Fasmarginal

2. Uppgifter ur tidigare tentamen



CHALMERS
Institutionen för signaler och system

TENTAMEN

KURSNAMN	Elektriska system för EI3
PROGRAM: namn åk / läsperiod	Elektroingenjör 180 hp 3/1
KURSBETECKNING	SSY010
EXAMINATOR	Erik Agrell
TID FÖR TENTAMEN	2014-10-28 kl 8:30-12:30
HJÄLPMEDEL	Typgodkänd räknare