# *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

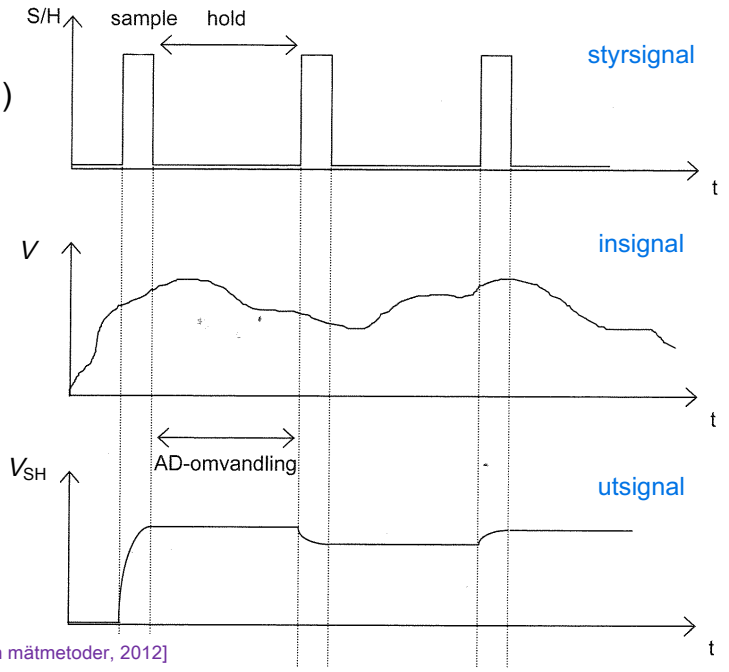
Bilder av E. Agrell



## *1. Mer om sample-and-hold*

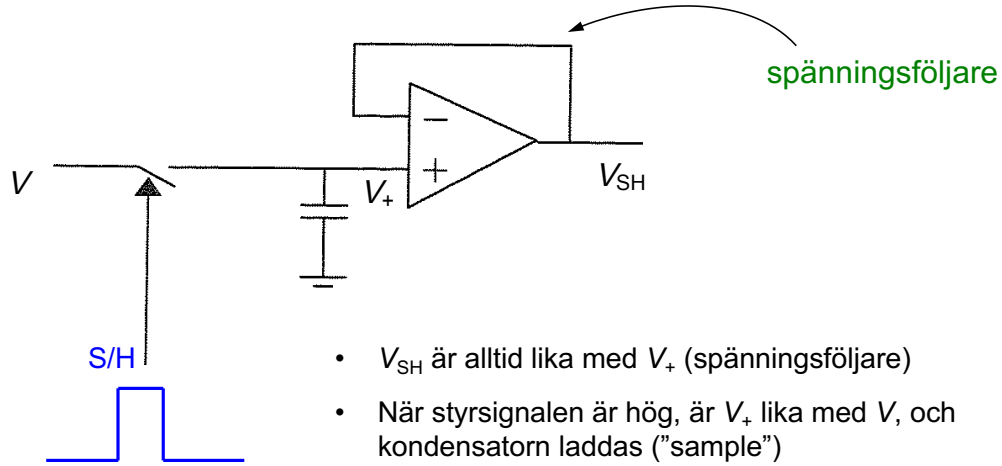
*Från F8:*

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




[Bengtsson: Elektriska mätsystem och mätmetoder, 2012]

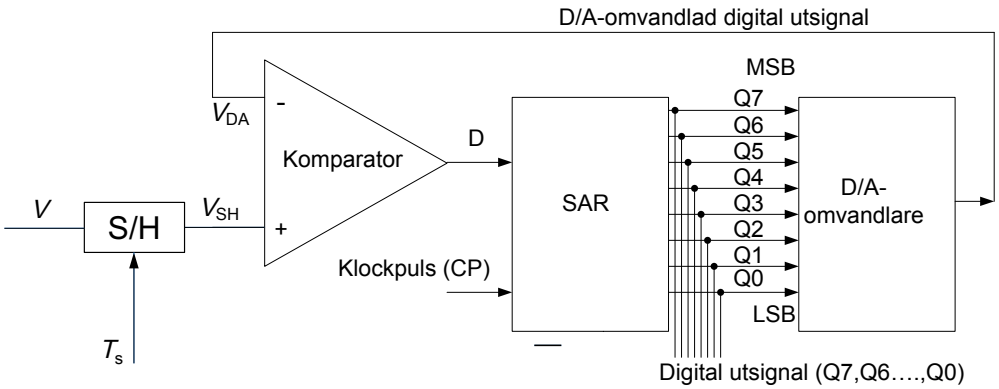
# Principen för sample-and-hold



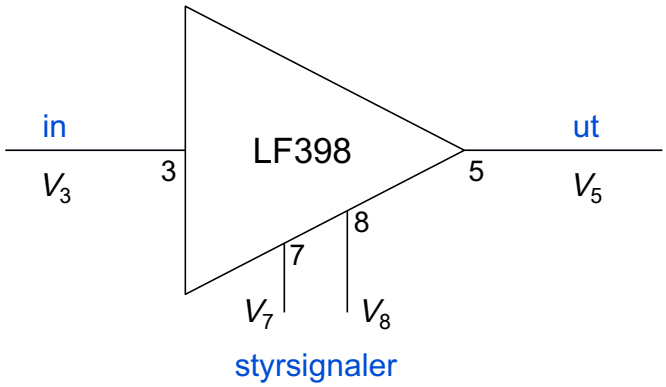

- $V_{SH}$ är alltid lika med $V_+$ (spänningsföljare)
- När styrsignalen är hög, är $V_+$ lika med $V$, och kondensatorn laddas (”sample”)
- När styrsignalen är låg, behåller kondensatorn spänningen $V_+$ (”hold”)



# Inkoppling till A/D-omvandlaren



# Styrning av S/H-kretsen LF398



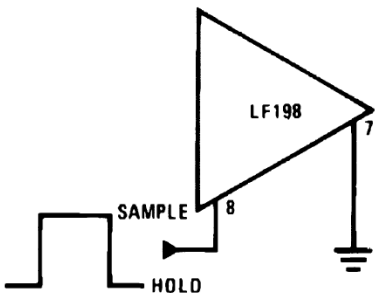

- Sample om $V_8 - V_7 > 1.4$ V $\Rightarrow V_5 = V_3$
- Hold om $V_8 - V_7 < 1.4$ V $\Rightarrow V_5$ behåller tidigare $V_3$

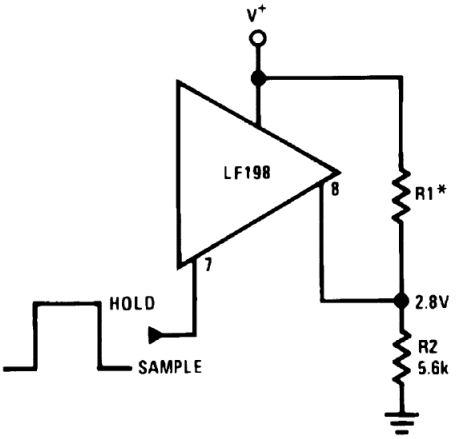


# Dimensionering av styrsignalen

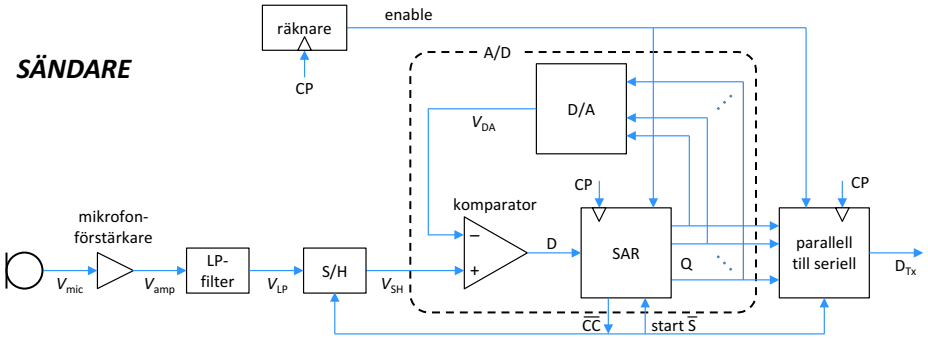
- Vill man sampla vid positiv puls, läggs styrsignalen på pinne 8
- Vill man sampla vid negativ puls, läggs styrsignalen på pinne 7
- Den andra pinnen ges en konstant referensspänning

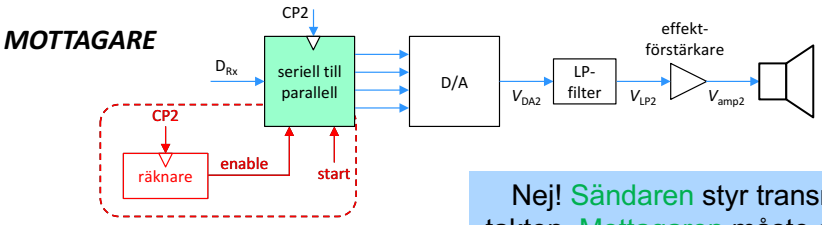
Exempel ur datablad:



# 2. Synkronisering i mottagare



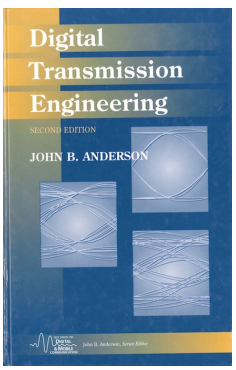

Funkar detta?

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.

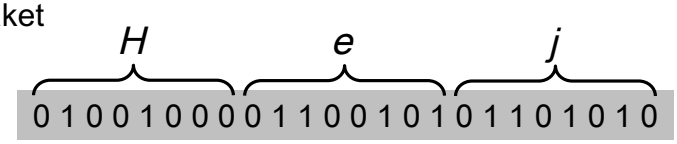
Probably the most underrated area in transmission design is synchronization. Little time is spent in most books on this subject, yet synchronization of various types probably consumes half of transmission design effort. In reality, there are several layers to the synchronization of a system: An RF system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



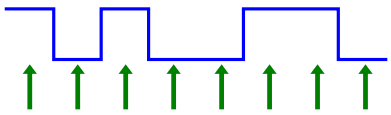

[John B. Anderson, *Digital Transmission Engineering*]

# Typer av synkronisering

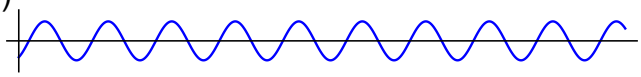
- Blocksynkronisering är att gruppera bitar till bytes och bytes till datapaket




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

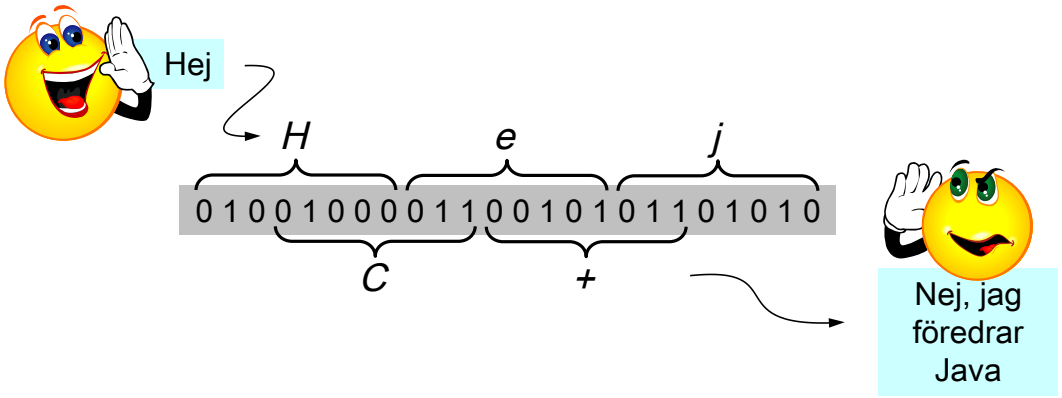


- Fassynkronisering är att identifiera fasläget hos en sinusvåg, vilket behövs om fasen bär information (fasmodulation, inte i denna kurs)



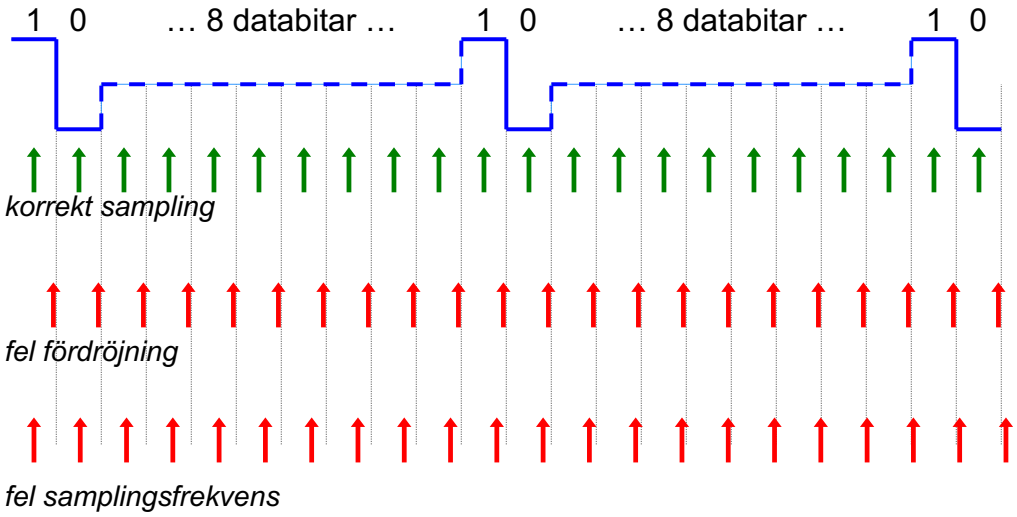


# Felaktig blocksynkronisering




*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?

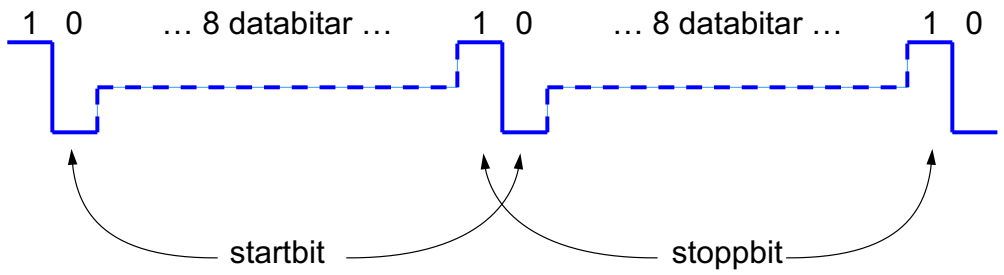
1. *Noggranna oscillatorer?* Nej, även bra kristalloscillatorer driver något.
2. *Använda sändarklockan även i mottagaren?* OK, men det kräver en sidokanal.
3. *Återvinna sändarklockan ur mottagen signal?* OK, om signalkvaliteten är hyfsad. ⇒ symbolsynkronisering

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

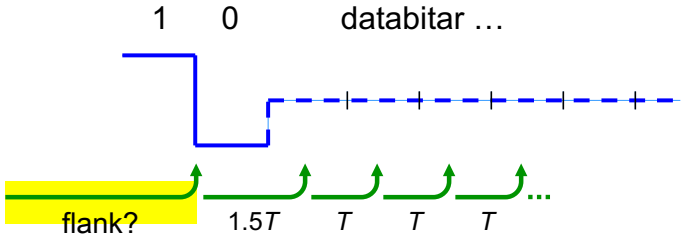
Klockåtervinning = ”clock recovery” = ”symbol synchronization”:

Att hitta rätt samplingstidpunkter för den mottagna signalen

*Grundprincip:*

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

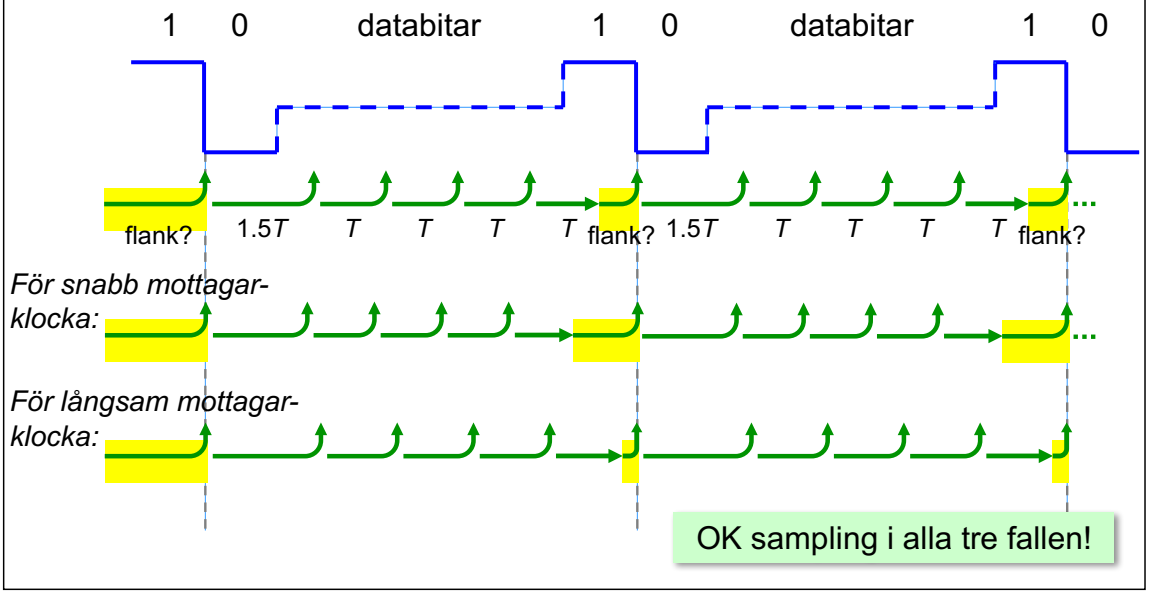
$T$ = symboltiden = 1/baudrate



# Klockdrift

Sändar- och mottagaroscillatorerna är inte helt synkrona

⇒ trigga på varje startbit

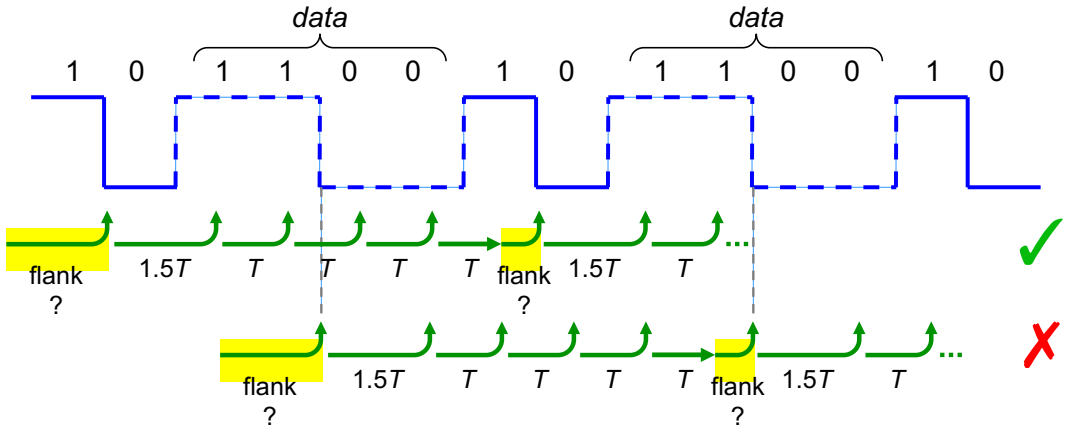



OK sampling i alla tre fallen!



# Problem: flanker i data

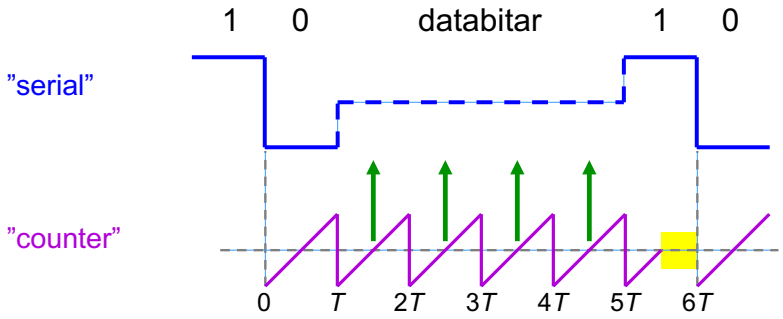
Regelbundna flanker i dataorden kan få mottagaren att trigga fel:




- 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

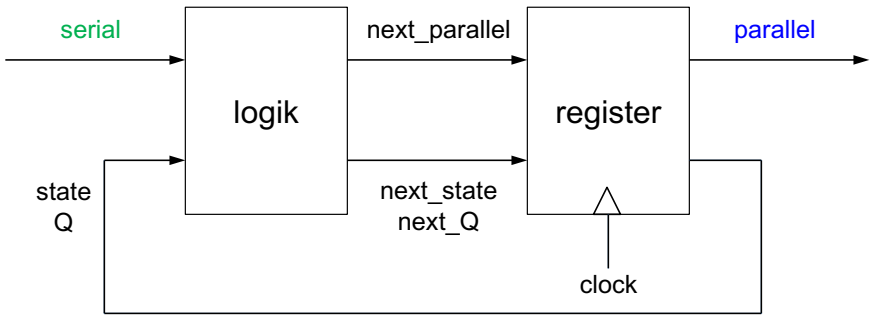
För att implementera en fördröjning behövs en klocka och en räknare



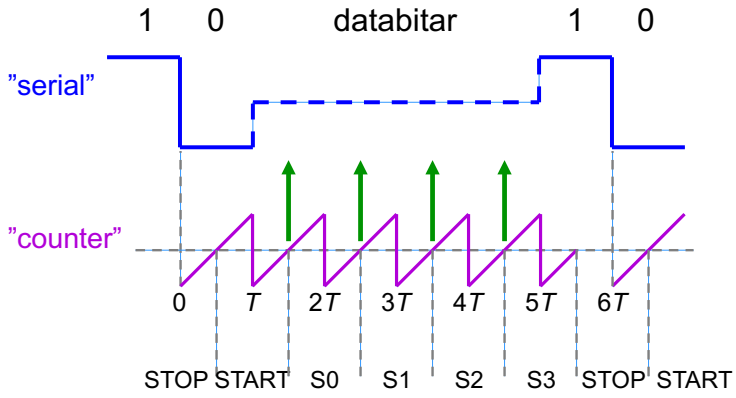

- Nollställ räknaren vid varje flank 1→0
- Nollställ räknaren när den når det värde som motsvarar *T*
- Sampla när räknaren är vid *halva* maxvärdet
- Börja vänta på nästa flank *T* tidsenheter efter sista datasamplingen
- Räkna bitar med hjälp av en tillståndsmaskin

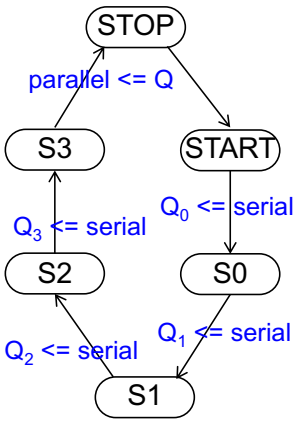


# Tillståndsmaskin enligt Mealy



Vi behöver ett tillstånd för varje bit



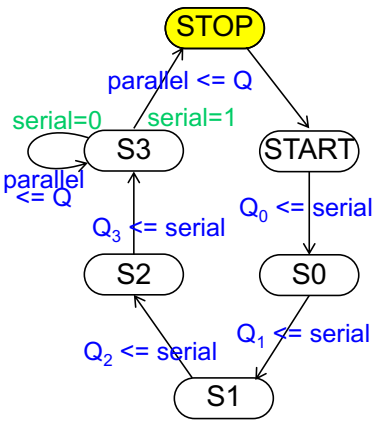




- Nytt tillstånd varje gång räknaren är vid halva maxvärdet
- Tillståndsmaskinen gör också serie-till-parallell-omvandling



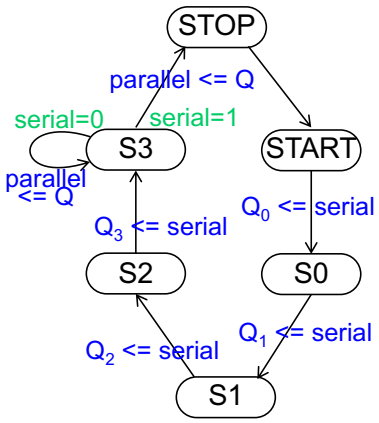
# Tillståndsmaskin med synkronisering

- 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



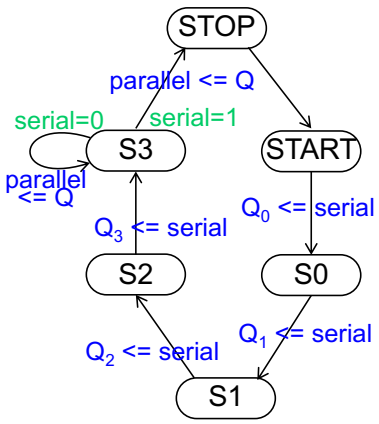
# Tillståndsmaskin i VHDL

```
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;
```





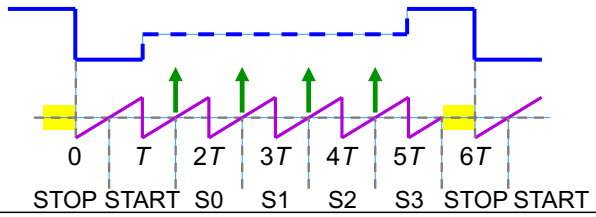

```
    when S2 =>
      next_state <= S3;
      next_Q(3) <= serial;
    when S3 =>
      next_parallel <= Q;
      if serial='0' then
        next_state <= S3;
      else
        next_state <= STOP;
      end if;
    when others =>   -- undefined state
      next_state <= STOP;
  end case;
end process;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

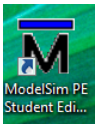
```
# header:
#vsim work.rx; view wave
#add wave clk50 reset serial state counter parallel

# 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));
```

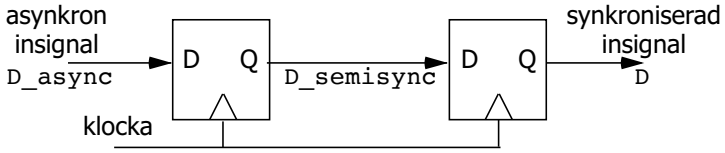
```
architecture state_machine of Rx is
  type StateType is (U,STOP,START,S0,S1,S2,S3);
  signal state, next_state: StateType;
  signal Q, next_Q, next_parallel: std_logic_vector(3 downto
0);
```

```
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