Design av mindre digitala system

Föreläsning
Digitalteknik, TSEA52
Mattias Krysander
Institutionen för systemteknik

LINKÖPINGS UNIVERSITET

Dagens föreläsning

- Kursinformation för HT2.
- Digitaltekniska byggblock
  - Introduktion av SR-vippa och register.
- Exempel på design av digitala system.
  - Sändare för asynkron seriell överföring
  - Ping-pong spel.
- Introduktion till laboration 4 (miniprojekt)

Kursinformation för HT2

Information om HT2, 2 hp

Kurshemsida:

http://www.isy.liu.se/edu/kurs/TSEA52/

Kursmoment

- Föreläsning 1-2 – lösning av större uppgifter

Lab/miniprojekt görs i par,

- Pass 1-3: 2h, anmälan
- Restpass 4h, drop-in

Kursinformation

- Modelsim och Xilinx finns i salarna:
  - Freja, Grinden
  - Ni ska ha access till Grinden.

Logga in och kör Modelsim och Xilinx på ixtab med studentkonto och Studentlösenord (Linux, Mac)

```
> ssh -Y LiU-ID@ixtab.edu.isy.liu.se
> module load mentor/modeltech10.4c
> vsim
> module load xilinx/14.7i
> ise
```

På Windows kan man ladda hem t ex mobaxterm för att få en ssh-client.

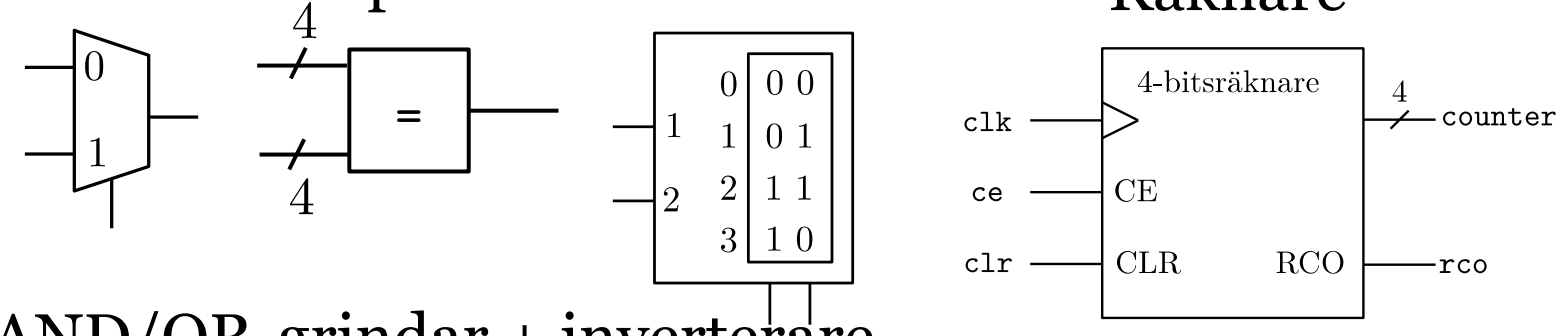
Handledare

- Petter Källström, petter.kallstrom@liu.se
- Narges Mohammadi Sarband, narges.mohammadi.sarband@liu.se

Digitaltekniska byggblock

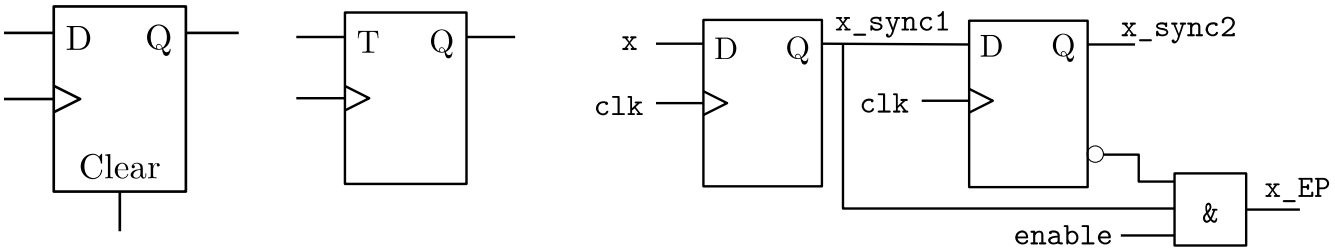
Digitaltekniska byggblock (använd dessa)

MUX Komparator ROM Räknare



AND/OR-grindar + inverterare

D-vippa T-vippa Enpulsare med synkronisering

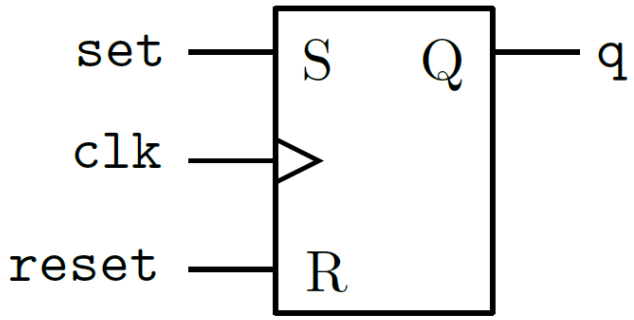


Komponenternas kod i pdf:en Digitaltekniska byggblock.

SR-vippan

Funktion

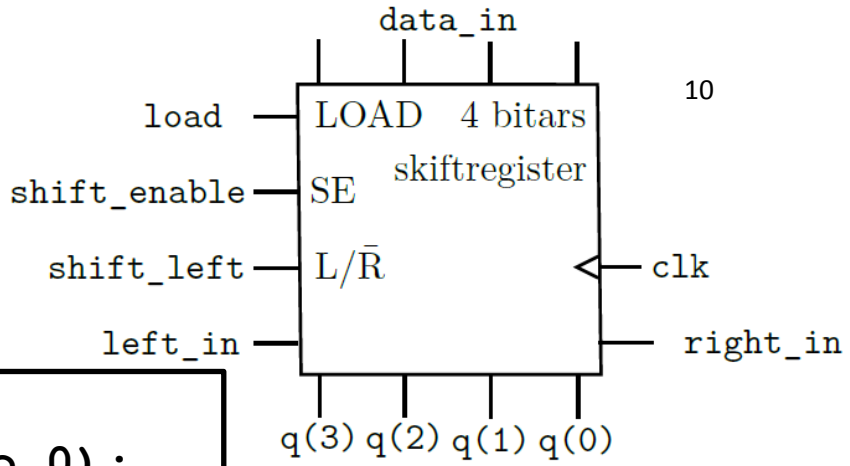
- set = reset = 0 => $q^+ = q$
- set = 1 => $q^+ = 1$
- reset = 1 => $q^+ = 0$




```
process(clk) begin
    if rising_edge(clk) then
        if (set = '1') then
            q <= '1';
        elsif (reset = '1') then
            q <= '0';
        end if;
    end if;
end process;
```

Dominerande set i exemplet:
set > reset

Skiftregister



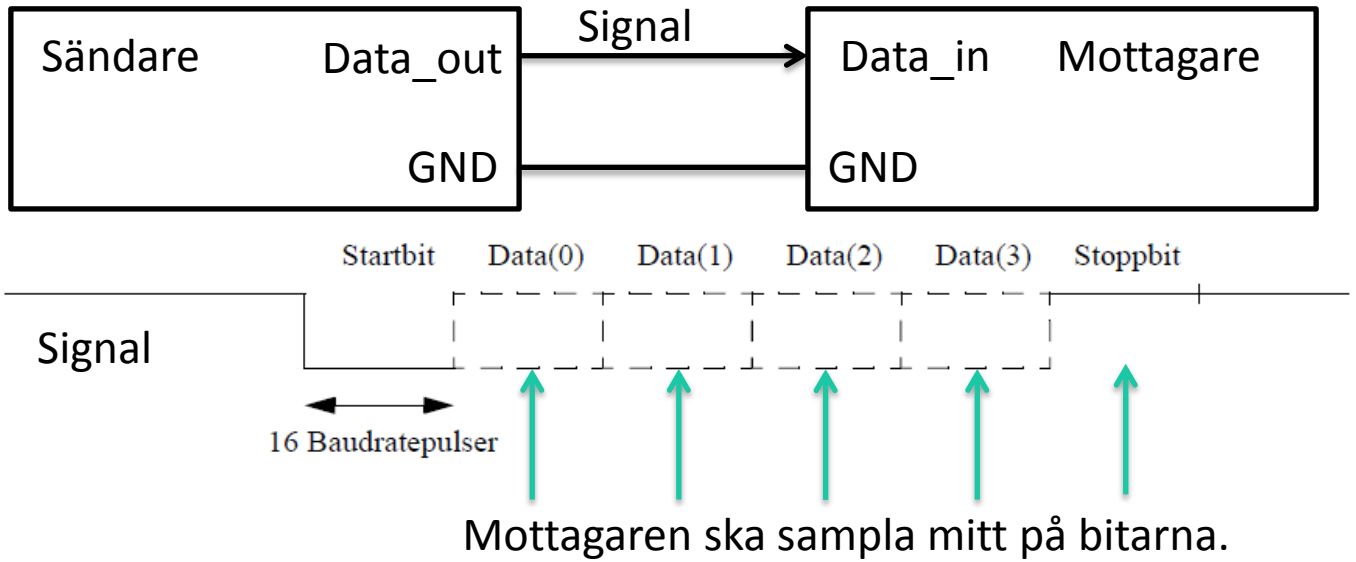

```
signal q, data_in :
       std_logic_vector(3 downto 0);

process(clk) begin
  if rising_edge(clk) then
    if (load = '1') then
      q <= data_in;
    elsif (shift_enable = '1') then
      if (shift_left = '1') then
        q <= q(2 downto 0) & right_in;
      else
        q <= left_in & q(3 downto 1);
      end if;
    end if;
  end if;
end process;
```

Dominerande load
LOAD>SE

Design av mindre digitala kretsar

Asynkron seriell kommunikation

Vid asynkron seriell överföring finns ingen gemensam klocka och bitarna skickas seriellt efter varandra.



Mottagaren ska sampla mitt på bitarna.

För korrekt mottagning måste mottagaren

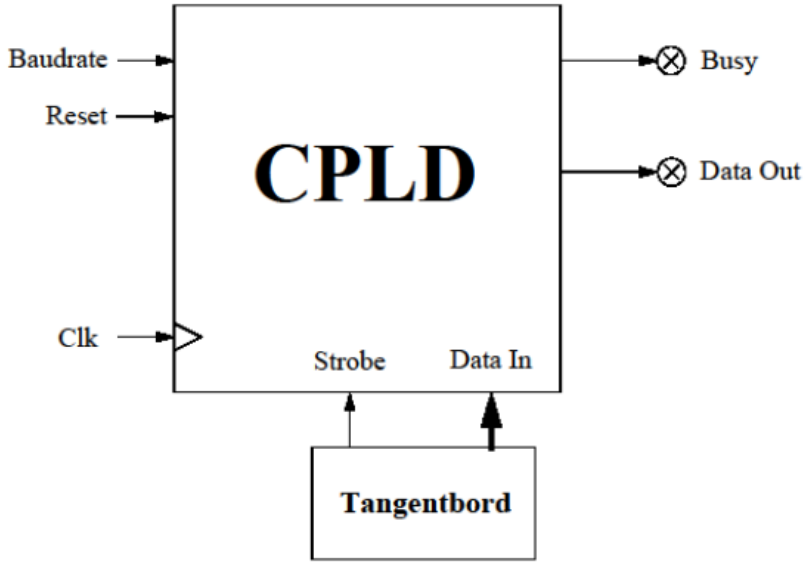
- Identifiera när startbiten börjar (fallande flank)
- Känna till datatakten på meddelandet (baudraten)

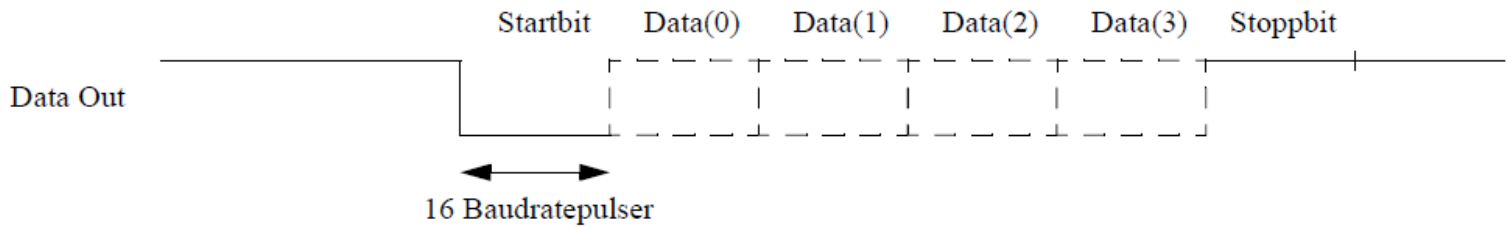
Sändare

Bygg en sändare som får information om fyra bitar från ett tangentbord och sänder ut detta seriellt.

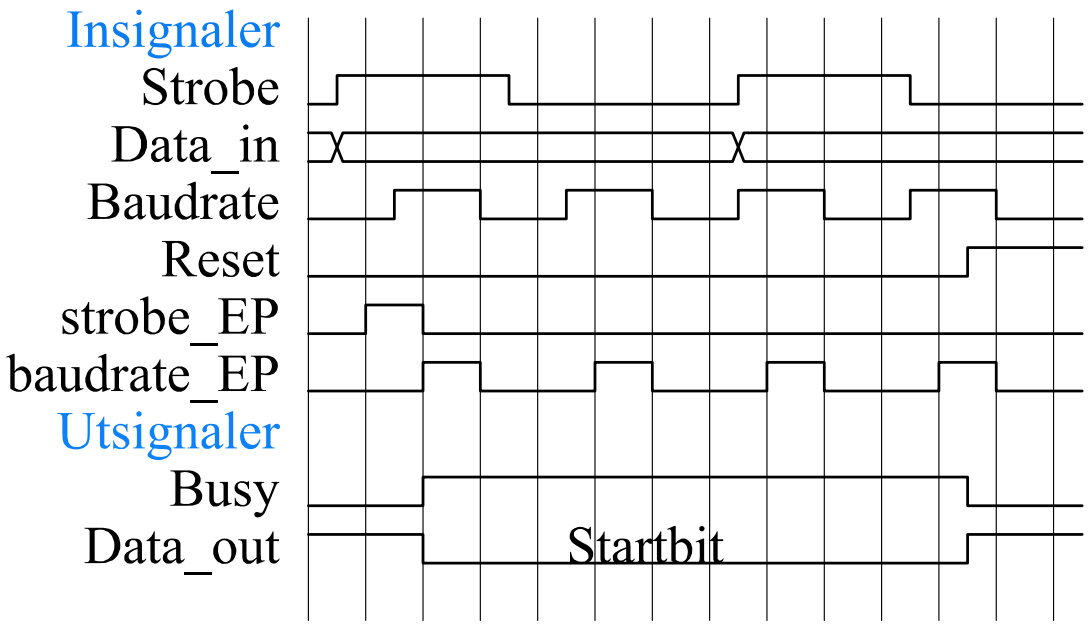
Formatet ska vara Startbit, databitar samt en stoppbit.

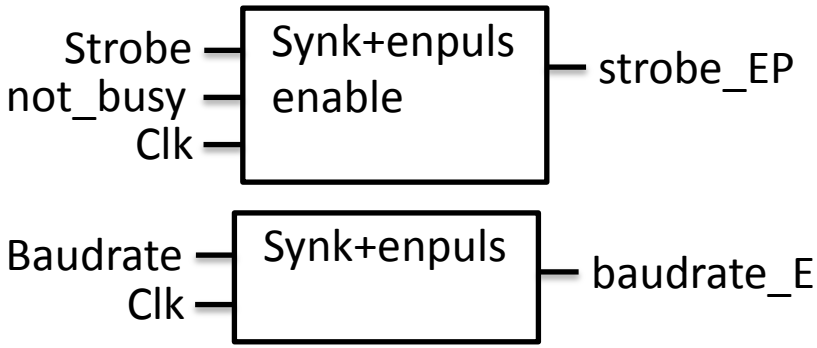
Pågående sändning ska ej avbrytas och indikeras med LED (Busy).

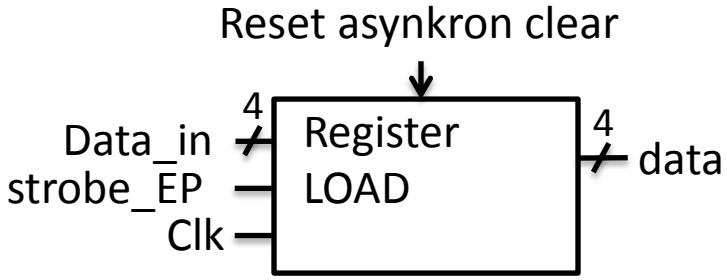




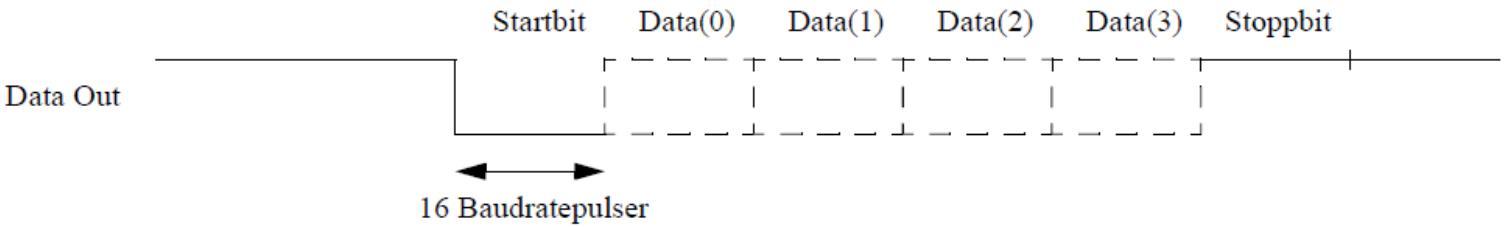
Insignaler


Insignaler
Strobe
Data_in
Baudrate
Reset
strobe_EP
baudrate_EP
Utsignaler
Busy
Data_out
Startbit


Strobe
not_busy
Clk
Synk+enpuls
enable
strobe_EP
Baudrate
Clk
Synk+enpuls
baudrate_EP

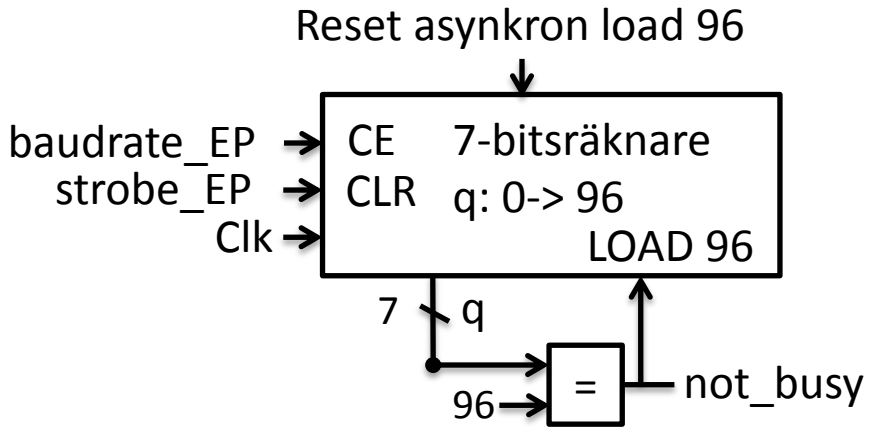

Reset asynkron clear
4
Data_in
strobe_EP
Clk
Register
LOAD
4
data

Sändning/tillstånd



Styr sändningen med en räknare som räknar antalet Baudratepulser q.

q	Data_out	Busy	$q_6q_5q_4q_3q_2q_1q_0$
0-15	Startbit	1	000 XXXX
16-31	Data(0)	1	001 XXXX
32-47	Data(1)	1	010 XXXX
48-63	Data(2)	1	011 XXXX
64-79	Data(3)	1	100 XXXX
80-95	Stoppbit	1	101 XXXX
96	1	0	110 0000

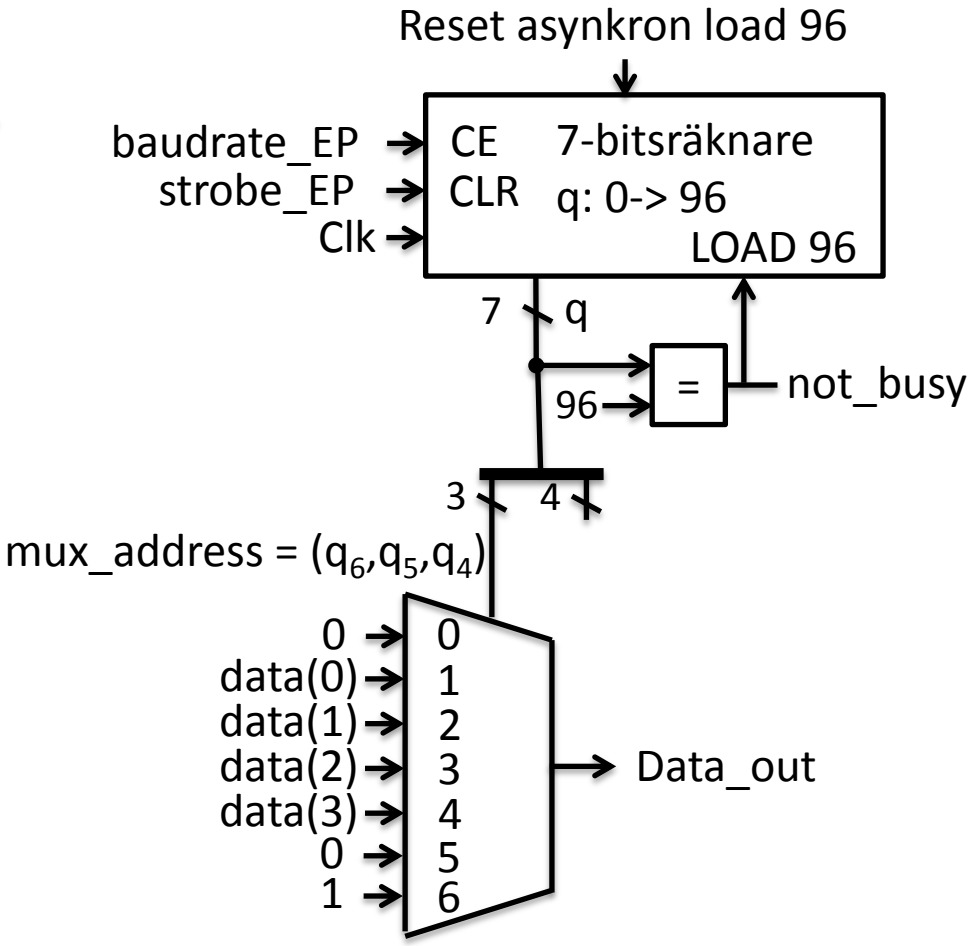


Prioritet: Asynkon LOAD > CLR > LOAD > CE

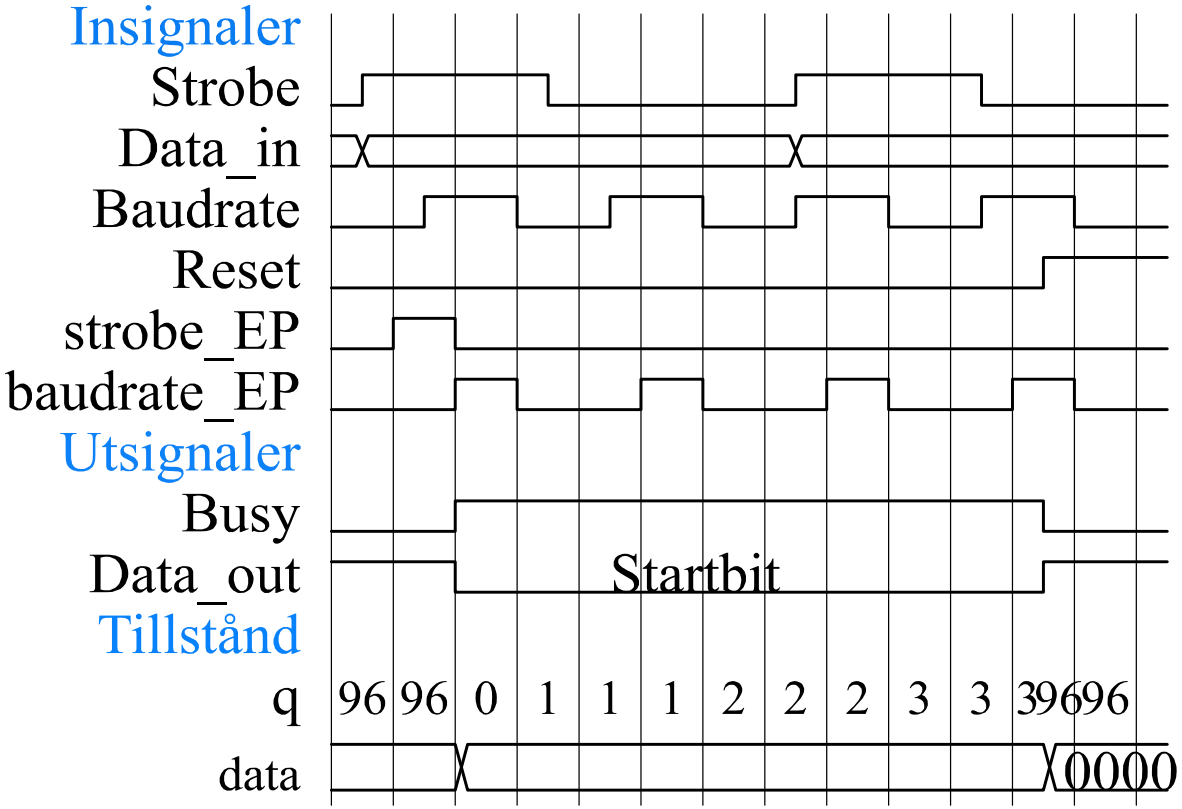
Utsignaler

q	Data_out	Busy	$q_6q_5q_4q_3q_2q_1q_0$	
0-15	Startbit=0	1	000	XXXX
16-31	data(0)	1	001	XXXX
32-47	data(1)	1	010	XXXX
48-63	data(2)	1	011	XXXX
64-79	data(3)	1	100	XXXX
80-95	Stoppbit=1	1	101	XXXX
96	1	0	110	0000

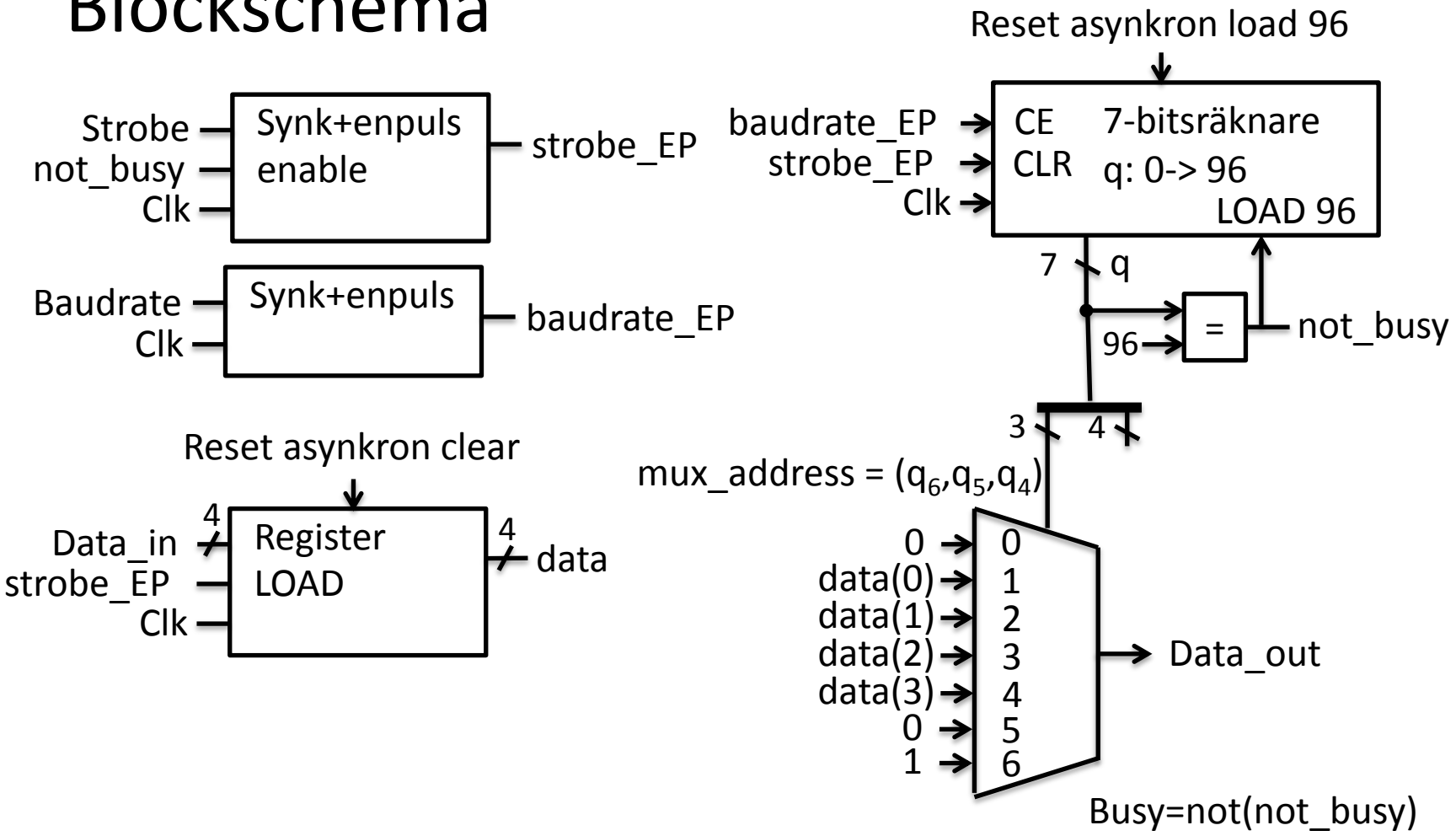
Busy=not(not_busy)



Analys

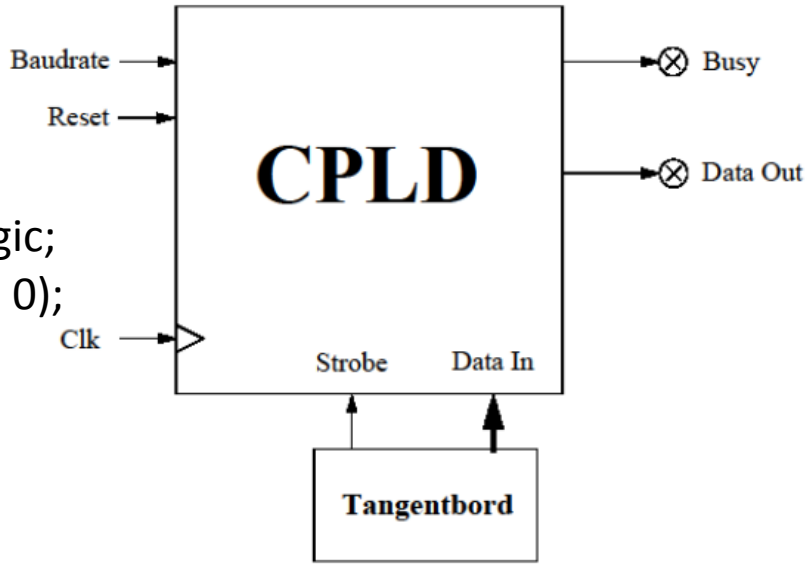

Insignaler
Strobe
Data_in
Baudrate
Reset
strobe_EP
baudrate_EP
Utsignaler
Busy
Data_out
Startbit
Tillstånd
q
96 96 0 1 1 1 2 2 2 3 3 39696
data
0000

Blockschema



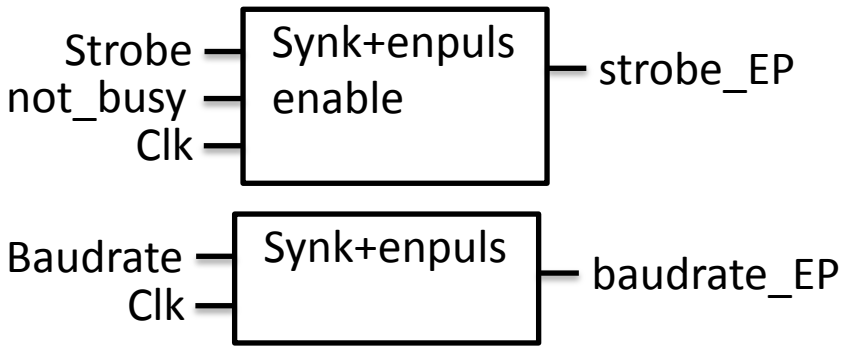
Översätt blockschema till VHDL-kod

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

entity transmitter is
  Port (Clk, Baudrate, Reset, Strobe : in std_logic;
        Data_in : in std_logic_vector(3 downto 0);
        Busy, Data_out : out std_logic);
end entity transmitter;

architecture Behavioral of transmitter is
signal …
begin
…
end architecture Behavioral;
```



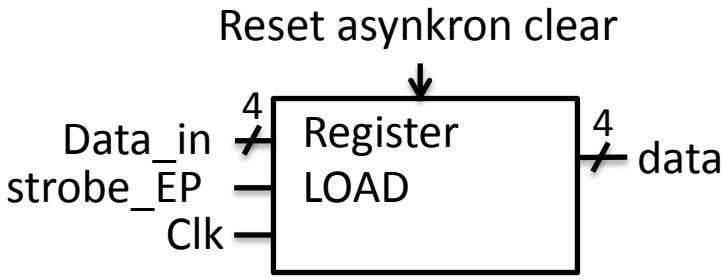
Insignaler

```
signal baudrate_sync1, baudrate_sync2, baudrate_EP : std_logic;
signal strobe_sync1, strobe_sync2, strobe_EP : std_logic;

process(Clk) begin
   if rising_edge(Clk) then
     baudrate_sync1 <= Baudrate;
     baudrate_sync2 <= baudrate_sync1;
     strobe_sync1 <= Strobe;
     strobe_sync2 <= strobe_sync1;
   end if;
 end process;
 baudrate_EP <= baudrate_sync1 and (not baudrate_sync2);
 strobe_EP <= strobe_sync1 and (not strobe_sync2) and not_busy;
```



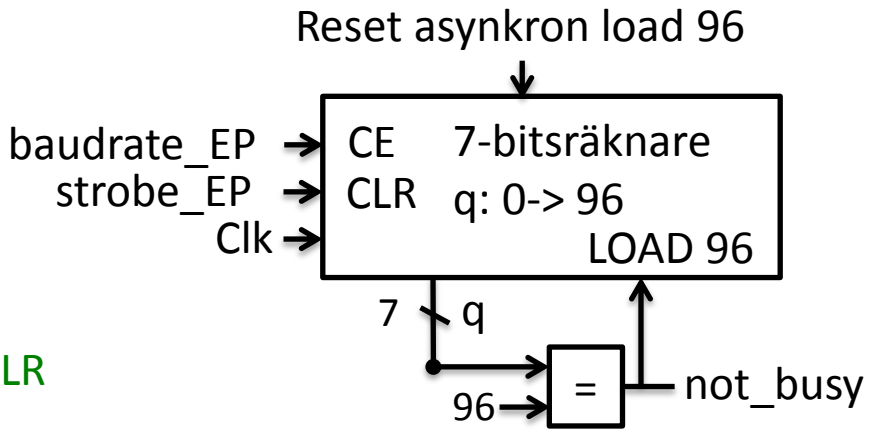
Indata

```
signal data : std_logic_vector(3 downto 0);

process(Clk,Reset) begin
   if (Reset = '1') then
      data <= "0000";
   elsif rising_edge(Clk) then
      if (strobe_EP = '1') then
         data <= Data_in;
      end if;
   end if;
end process;
```



Räknare

```
signal q : unsigned(6 downto 0);
signal not_busy : std_logic;

process(Clk,Reset) begin
   if (Reset = '1') then
      q <= "1100000";
   elsif rising_edge(Clk) then
      if (strobe_EP = '1') then          -- CLR
        q <= "0000000";
      elsif (not_busy = '1') then        -- LOAD 96
        q <= "1100000";                  -- to_unsigned(96,7)
      elsif (baudrate_EP = '1') then -- CE
        q <= q + 1;
      end if;
   end if;
 end process;
 not_busy <= '1' when (q = 96) else '0';
```

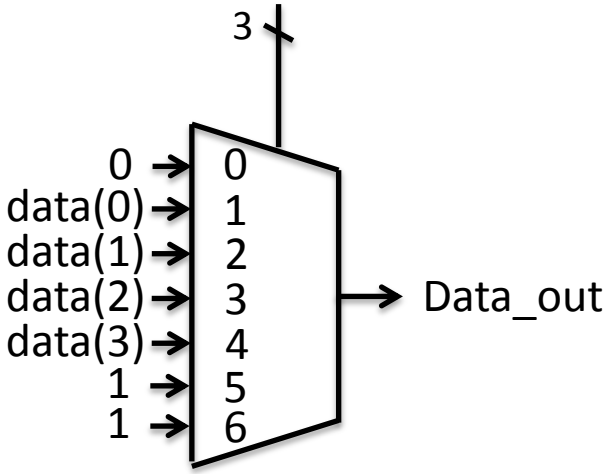


Prioritet: Asynkon LOAD > CLR > LOAD > CE

Utsignaler

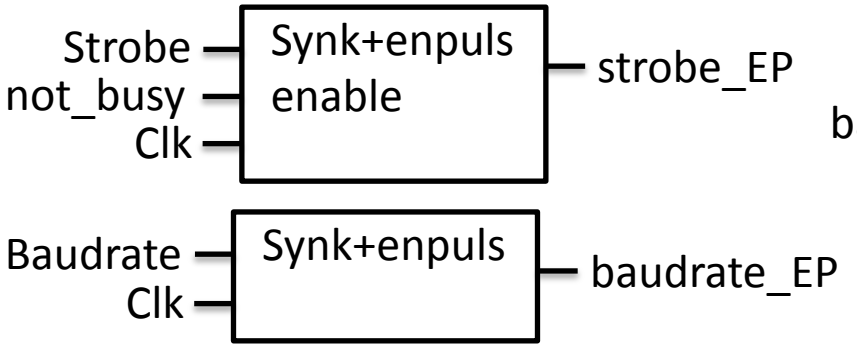
```
signal mux_address : std_logic_vector(2 downto 0);

mux_address <= std_logic_vector(q(6 downto 4));

with mux_address select
   Data_out <= '0'    when "000",   -- Startbit
               data(0) when "001",   -- Databit 0
               data(1) when "010",   -- Databit 1
               data(2) when "011",   -- Databit 2
               data(3) when "100",   -- Databit 3
                   '1' when "101",   -- Stoppbit
                   '1' when others;  -- Idle (not busy)

Busy <= not (not_busy);
```

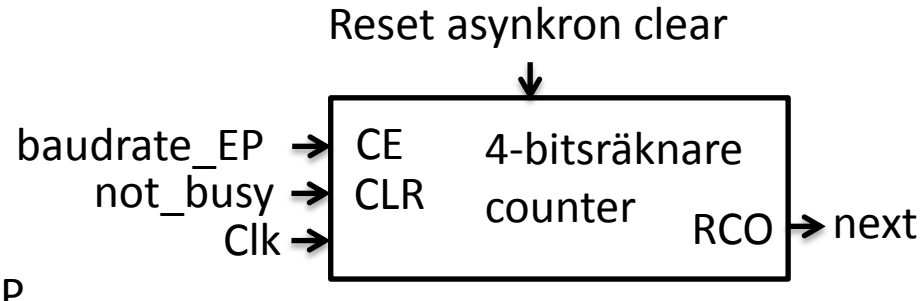
mux_address = (q_6,q_5,q_4)

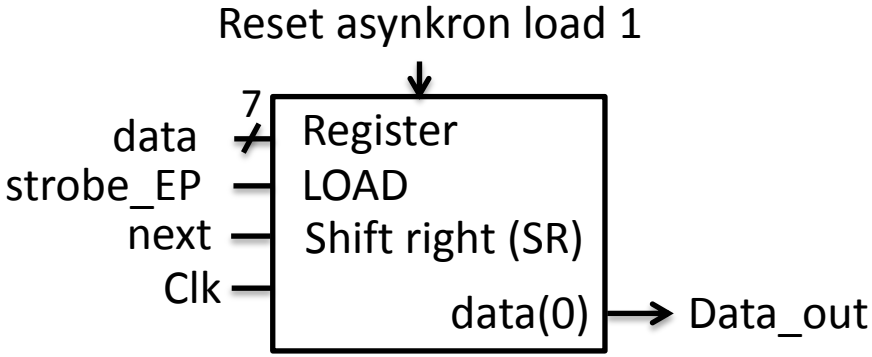


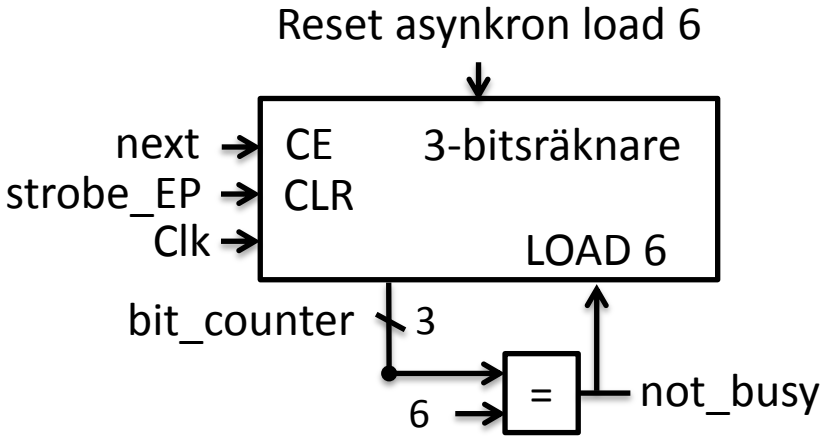
Busy=not(not_busy)

Alternativ med skiftregister









data(6 downto 0) = 1 & 1 & Data_in & 0;

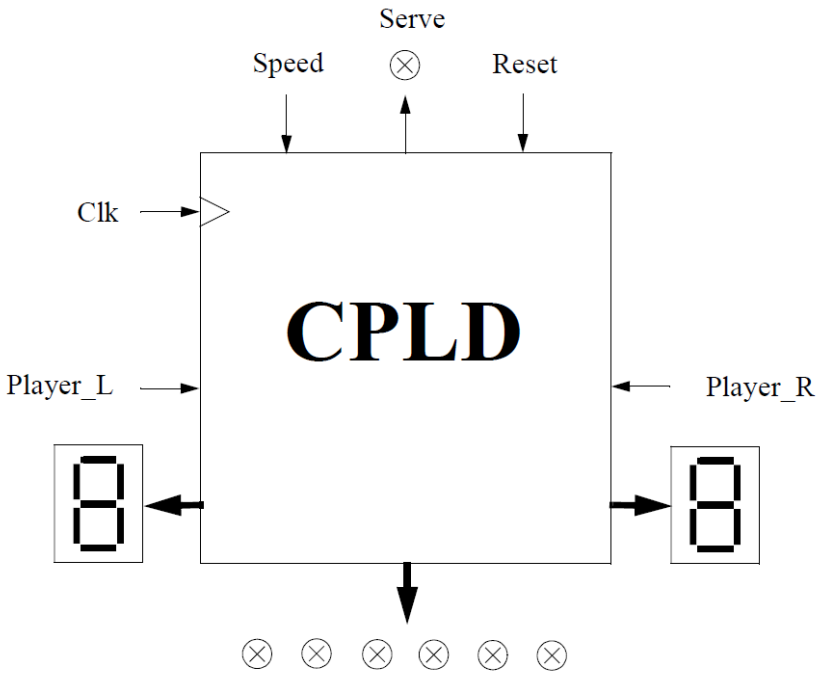
Busy=not(not_busy)

Ping-pong spel

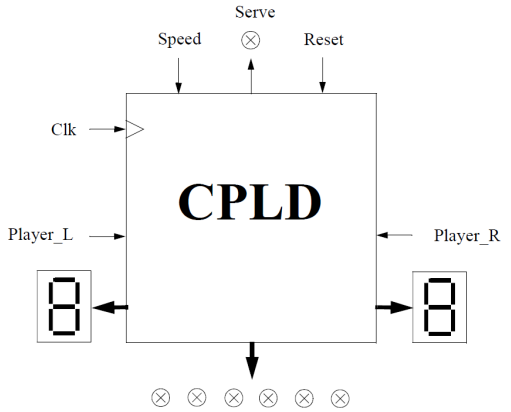
Ping-pong spel

En ”Boll”, markerad med en tänd lysdiod, ska förflyttas fram och tillbaka över en spelplan bestående av en rad lysdioder.

Spelet fortgår så länge de bägge spelarna trycker ned sin knapp exakt då bollen befinner sig i respektive ändläge.

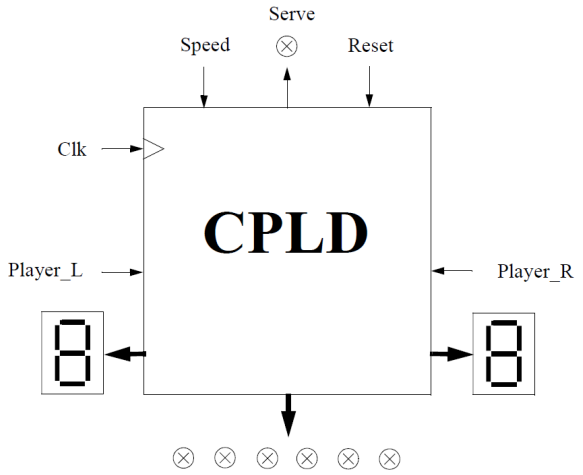


Spelregler

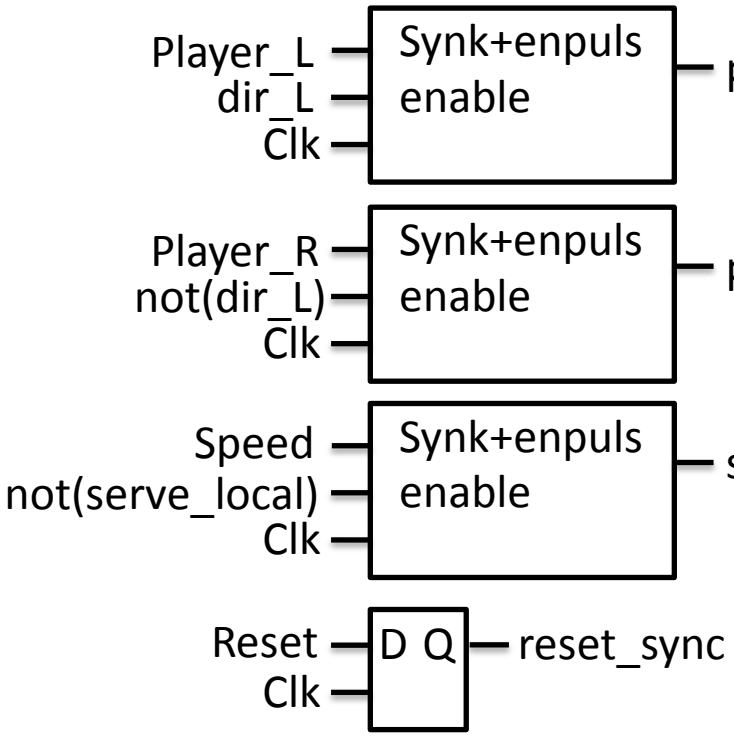


1. Den vänstra spelaren börjar att serva vid spelets start.
2. Serve markeras med stillastående boll på den servandes sida, samt med en LED (Serve).
3. Bollen börjar att röra sig då knappen trycks ned.
4. Om en spelare trycker ner knappen exakt när bollen är i ändläget börjar bollen röra sig i andra riktningen och det blir den andra spelarens tur.
5. Om en spelare trycker för tidigt eller om bollen går ut, vinner den andre bollen.
6. Den som vinner en boll får ett poäng och får serva.
7. Knapptryckningar av spelaren som inte är i tur ska ignoreras.
8. Insignalen Speed ska bestämma hur fort bollen rör sig.

Insignaler



Låt **dir_L** vara ett tillstånd så att:
dir_L = 1: bollen går åt vänster, vänster spelares tur
dir_L = 0: bollen går åt höger, höger spelares tur

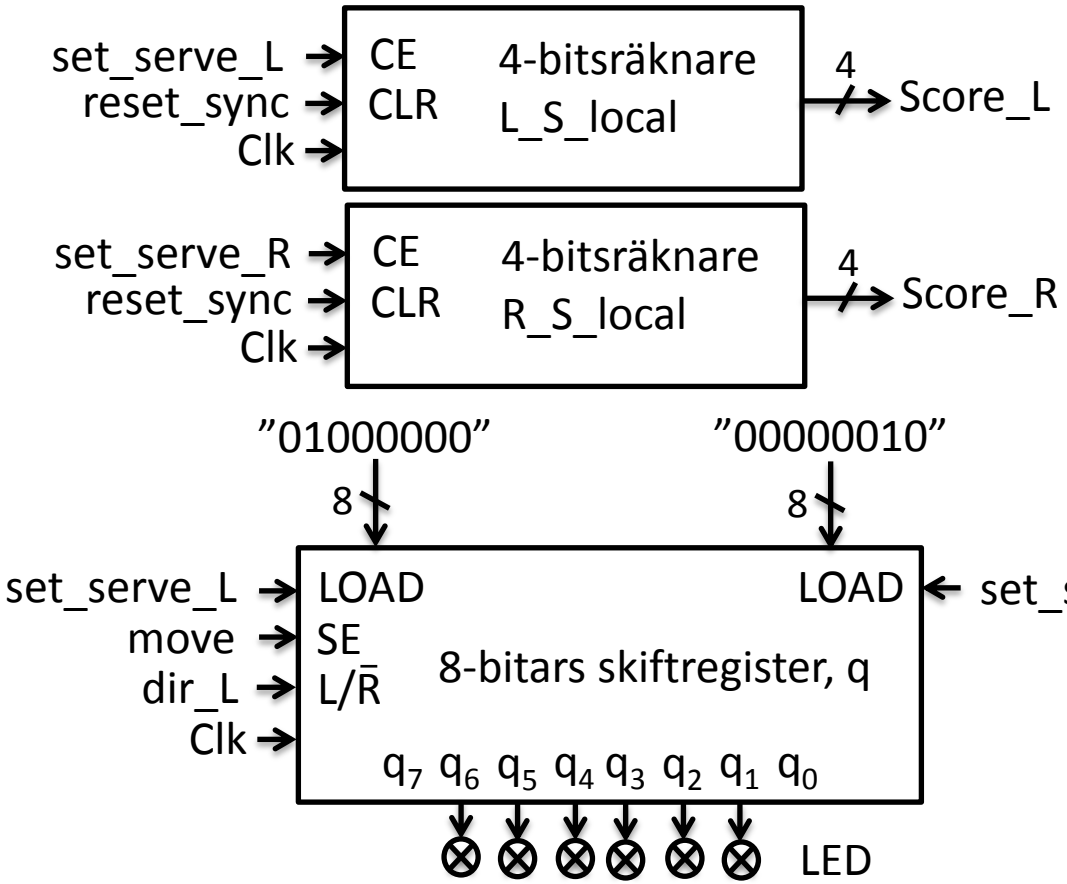


Player_L ska bara kunna påverka spelet när det är dennes tur, dvs då dir_L = 1.

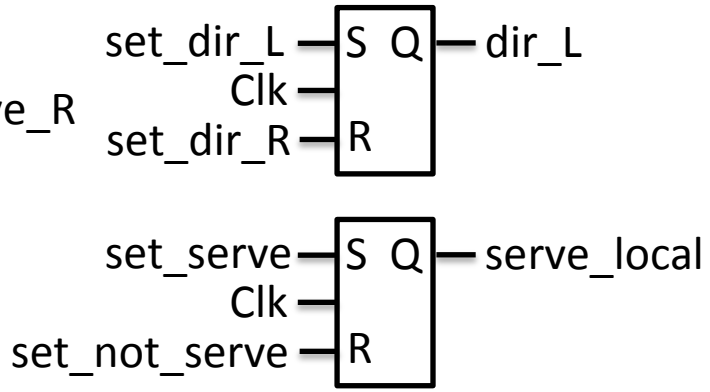
Player_R ska bara kunna påverka spelet när det är dennes tur, dvs då dir_L = 0.

speed_EP styr när bollen ska flytta ett steg
Bollen rör sig bara när det inte är serve.
Låt **serve_local** vara en intern signal = Serve.

Tillstånd

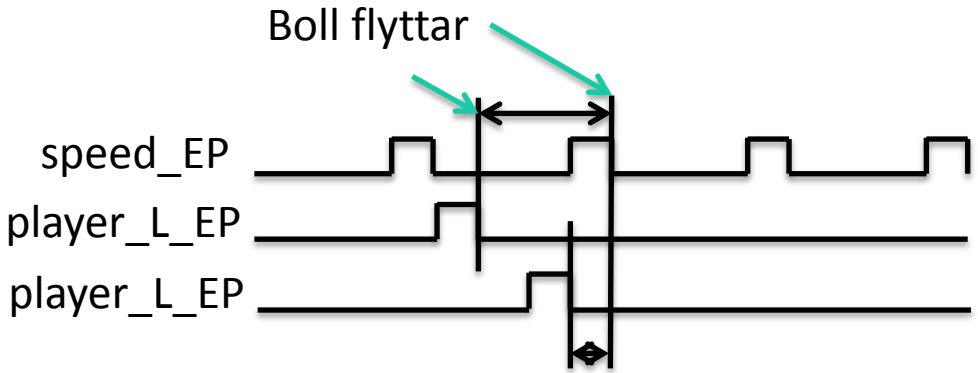


- När en spelare vinner en boll får den serven och en poäng. Låt **set_serve_L/R** initiera serve.
- Inför en signal **move** som initierar en bollförflyttning.



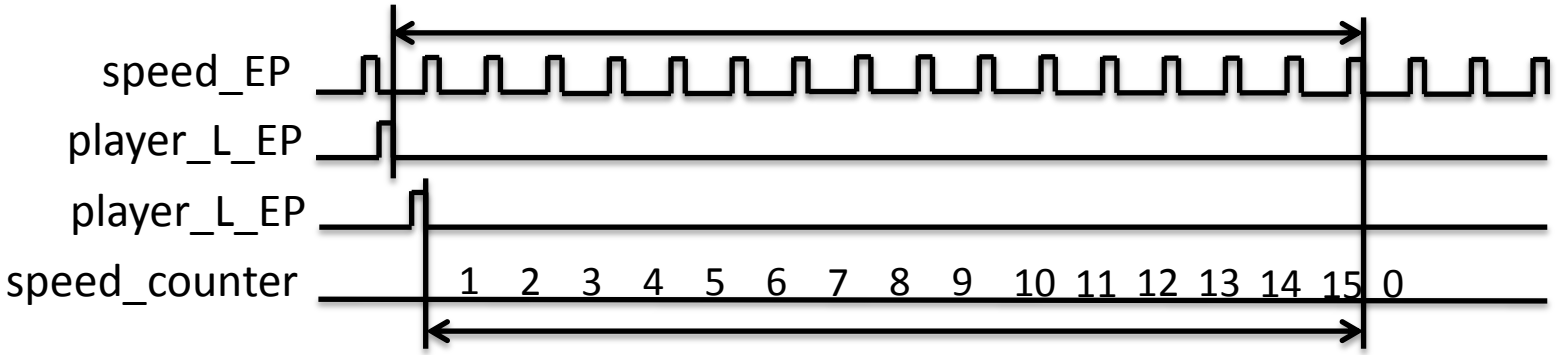
Speed

Problem:



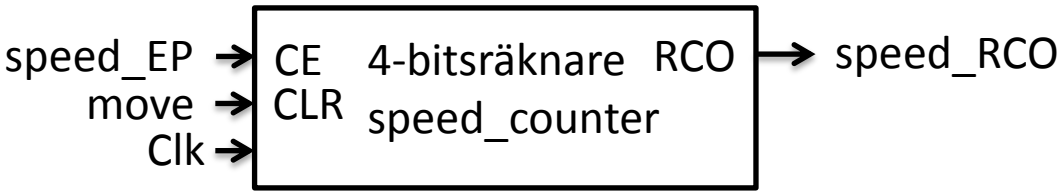
Bollen hoppar vidare olika snabbt beroende på när knappen trycks ner i förhållande till när speed_EP-pulserna kommer.

Lösning: Räkna 16 speedpulser innan förflyttning.

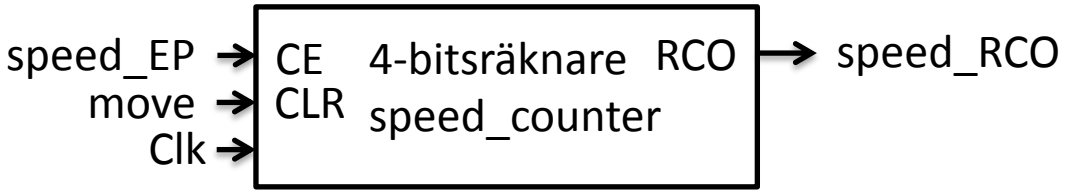


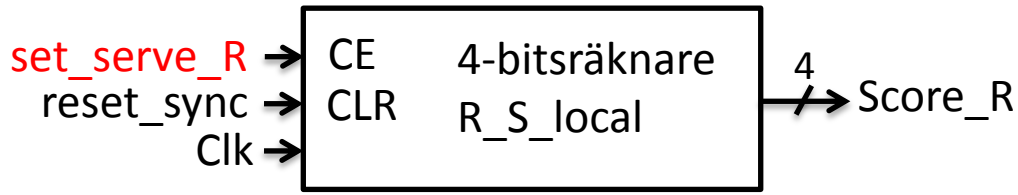
Nu hoppar bollen vidare i princip lika fort.

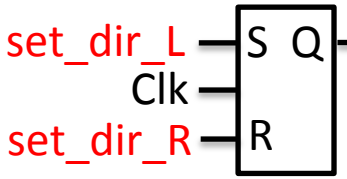
Speed_counter

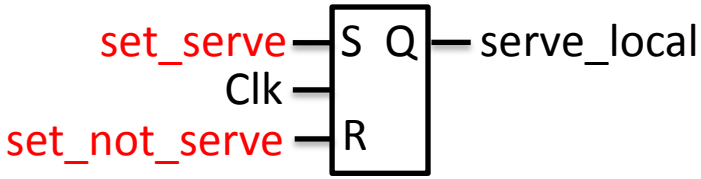


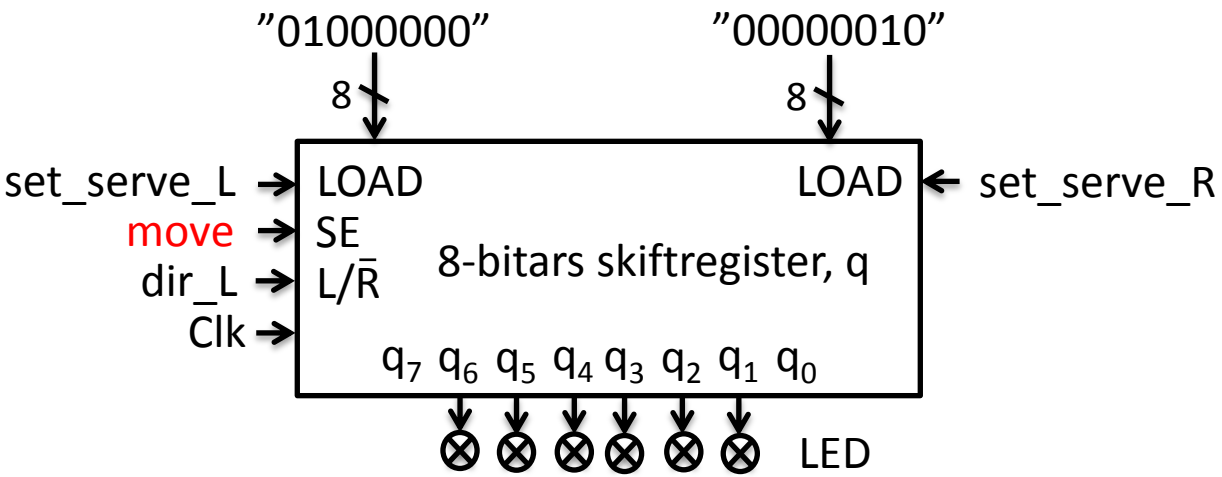
Styrsignaler











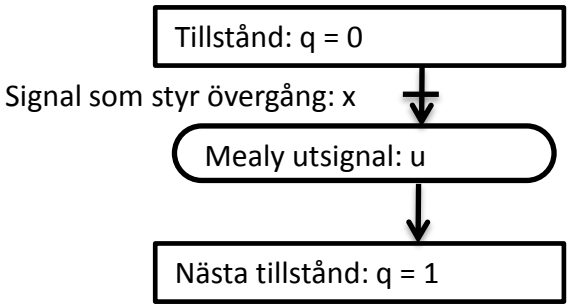
Styrsignaler:
set_serve_L
set_serve_R
set_serve
set_not_serve
move
set_dir_L
set_dir_R

Flödesdiagram

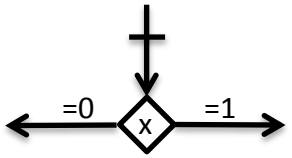
När det finns för många scenarion för att rita tidsdiagram kan flödesdiagram vara ett alternativ.

Tolkning

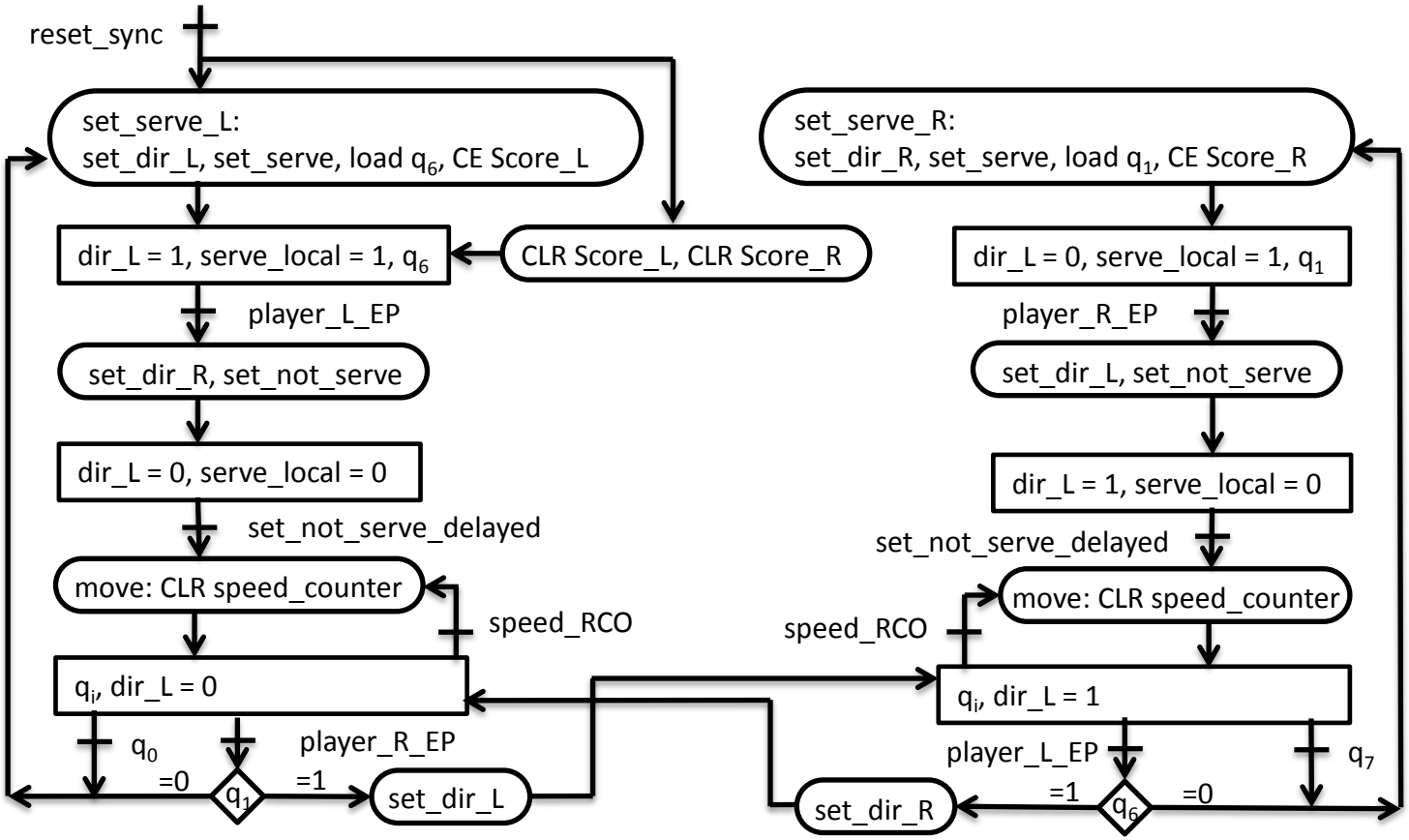
Om vi är i tillstånd q=0 och signalen x blir 1 så blir signalen u = 1 i samma klockintervall och i nästa klockintervall blir q=1



Om x = 0 följ den vänstra pilen om x=1 följ den högra pilen.

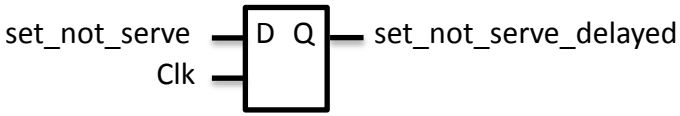


Programflöde och styrsignaluttryck

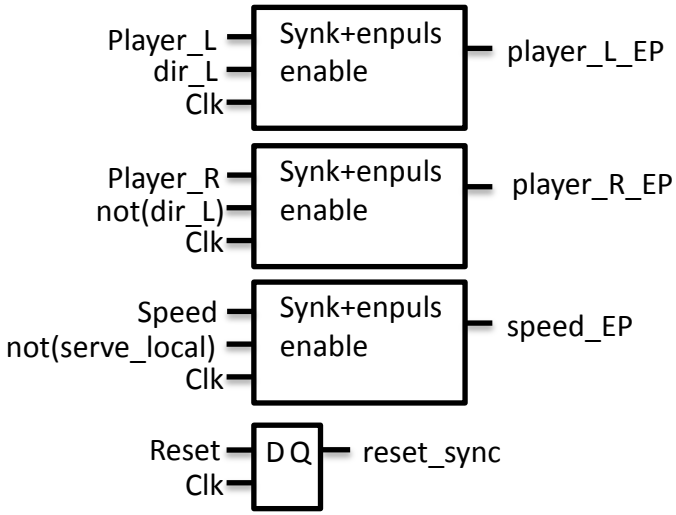


set_serve_L = reset_sync or q_0 or (player_R_EP and (not q_1))
set_serve = set_serve_L or set_serve_R
move = set_not_serve_delayed or speed_RCO
set_dir_L = set_serve_L or player_R_EP
set_dir_R = set_serve_R or player_L_EP

set_serve_R = q_7 or (player_L_EP and (not q_6))
set_not_serve = serve_local and (player_L_EP or player_R_EP)

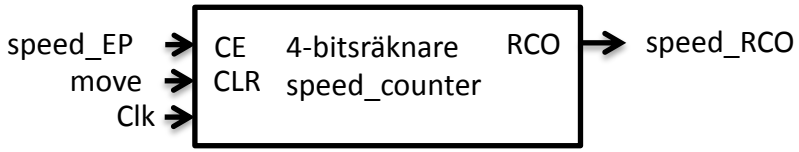


Insignaler




```
process(Clk) begin
   if rising_edge(Clk) then
      player_L_sync1 <= Player_L;
      player_L_sync2 <= player_L_sync1;
      player_R_sync1 <= Player_R;
      player_R_sync2 <= player_R_sync1;
      speed_sync1 <= Speed;
      speed_sync2 <= speed_sync1;
      reset_sync <= Reset;
   end if;
end process;
player_L_EP <= player_L_sync1 and (not player_L_sync2) and dir_L;
player_R_EP <= player_R_sync1 and (not player_R_sync2) and not(dir_L);
speed_EP <= speed_sync1 and (not speed_sync2) and (not serve_local);
```

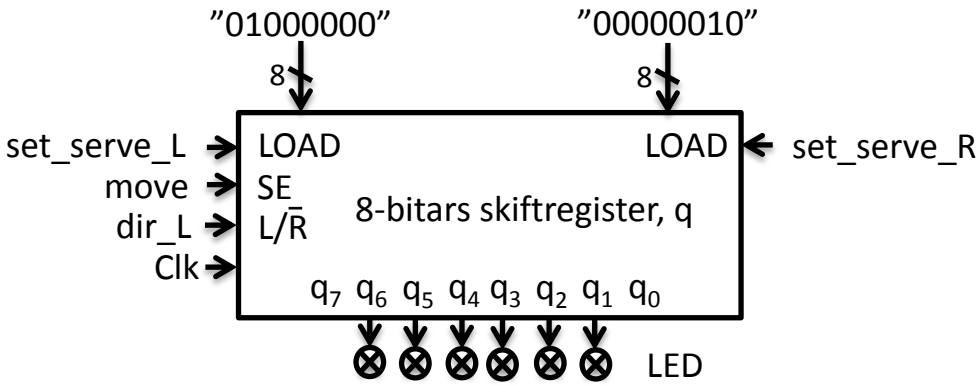
Speed counter




```
process(Clk) begin
   if rising_edge(Clk) then
      if (move = '1') then
         speed_counter <= "0000";
      elsif (speed_EP = '1') then
         speed_counter <= speed_counter + 1;
      end if;
   end if;
end process;
speed_RCO <= '1' when ((speed_EP = '1') and (speed_counter = 15))
                  else '0';
```

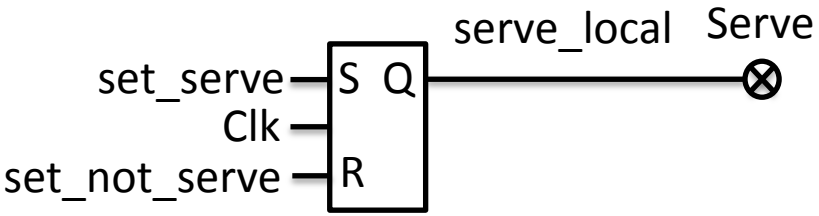
Skiftregistret

```
process(Clk) begin
   if rising_edge(Clk) then
      if (set_serve_L = '1') then
         q <= "01000000";
      elsif (set_serve_R = '1') then
         q <= "00000010";
      elsif (move = '1') then
         if (dir_L = '1') then
            q <= q(6 downto 0) & '0';
         else
            q <= '0' & q(7 downto 1);
         end if;
      end if;
   end if;
end process;
LED <= q(6 downto 1);
```

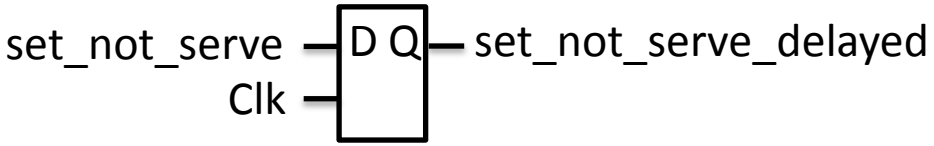


Serve

```
process(Clk) begin
   if rising_edge(Clk) then
      if (set_serve = '1') then
         serve_local <= '1';
      elsif (set_not_serve = '1') then
         serve_local <= '0';
      end if;
   end if;
end process;
Serve <= serve_local;
```

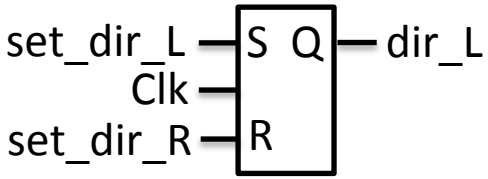



```
process(Clk) begin
   if rising_edge(Clk) then
      set_not_serve_delayed <= set_not_serve;
   end if;
end process;
```

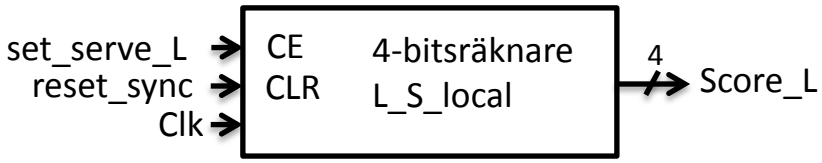


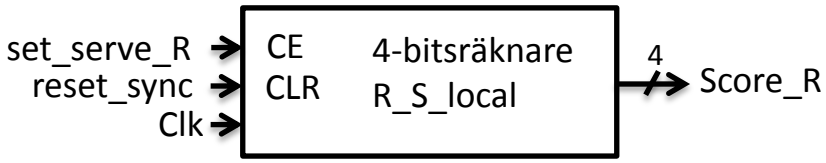
Direction

```
process(Clk) begin
   if rising_edge(Clk) then
      if (set_dir_L = '1') then
         dir_L <= '1';
      elsif (set_dir_R = '1') then
         dir_L <= '0';
       end if;
   end if;
end process;
```



Score






```
process(Clk) begin
   if rising_edge(Clk) then
      if (reset_sync = '1') then
         L_S_local <= "0000";
      elsif (set_serve_L = '1') then
         L_S_local <= L_S_local + 1;
      end if;
   end if;
end process;
Score_L <= std_logic_vector(L_S_local);
```

```
process(Clk) begin
   if rising_edge(Clk) then
      if (reset_sync = '1') then
         R_S_local <= "0000";
      elsif (set_serve_R = '1') then
         R_S_local <= R_S_local + 1;
      end if;
   end if;
end process;
Score_R <= std_logic_vector(R_S_local);
```

Styrsignaler

set_serve_L = reset_sync or q_0 or (player_R_EP and (not q_1))
set_serve_R = q_7 or (player_L_EP and (not q_6))

set_serve = set_serve_L or set_serve_R
set_not_serve = serve_local and (player_L_EP or player_R_EP)

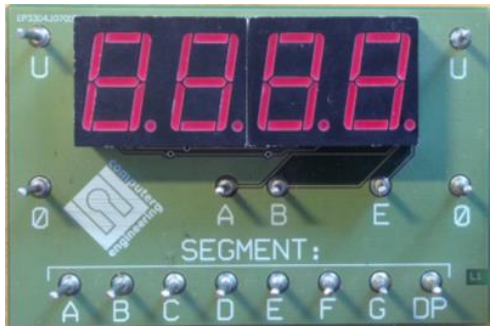
move = set_not_serve_delayed or speed_RCO

set_dir_L = set_serve_L or player_R_EP
set_dir_R = set_serve_R or player_L_EP

```
set_serve_L <= reset_sync or q(0) or (player_R_EP and (not q(1)));
set_serve_R <= q(7) or (player_L_EP and (not q(6)));

set_serve <= set_serve_L or set_serve_R;
set_not_serve <= serve_local and (player_R_EP or player_L_EP);

move <= set_not_serve_delayed or speed_RCO;

set_dir_L <= player_R_EP or set_serve_L;
set_dir_R <= player_L_EP or set_serve_R;
```

Lab 4/miniprojekt

Lab 4



- Bland annat konstruera ett digitalt tidtagarur.
- Olika klockor
  - Systemklocka 8 MHz
  - Klocka som styr uppdateringsfrekvens på display (variabel frekvens)
  - Klocka som styr tidräkning 100 Hz
- Synkronisera klockpulser med systemklockan:
  - synkronisering + enpulsning

Examination

- Vid examination ska logiskt blockschema, VHDL-kod samt korrekt fungerande krets uppvisas.
- Logiskt blockschema med block enligt dokumentet digitaltekniska byggblock och enklare logiska uttryck.
  - Egendesignade sekvenskretsar förutom krets också dokumenteras med tillståndsdiagram.
- VHDL-kod
  - Varje block ska motsvara ett kodavsnitt, t ex en process-sats/sekvenskrets
  - Överensstämmande signalnamn i blockschema och kod.

Lycka till med den avslutande laborationen!
Digitalteknik
Mattias Krysander

www.liu.se

LINKÖPINGS UNIVERSITET