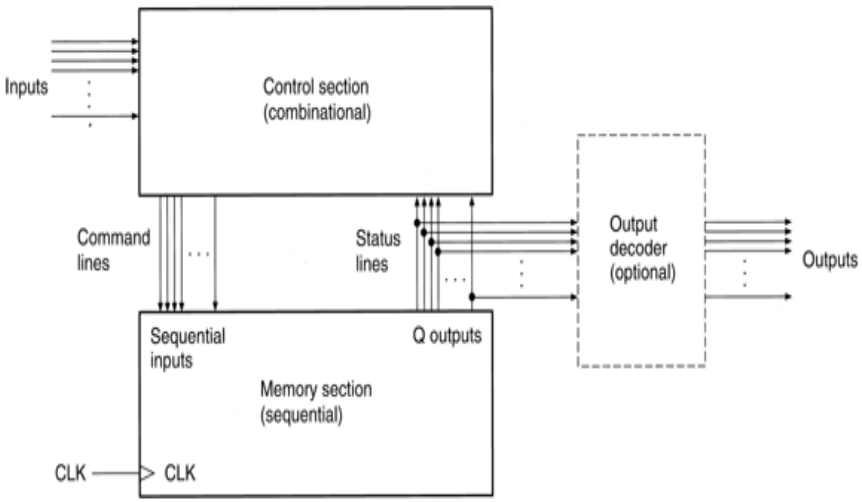
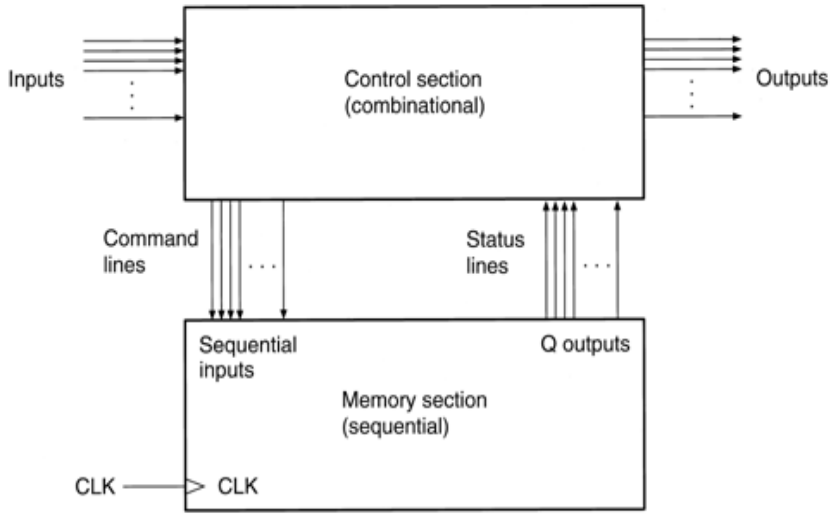
F20_000 Tilstandsmaskiner

Tilstandsmaskiner deler vi i to grupper

Moor-maskin

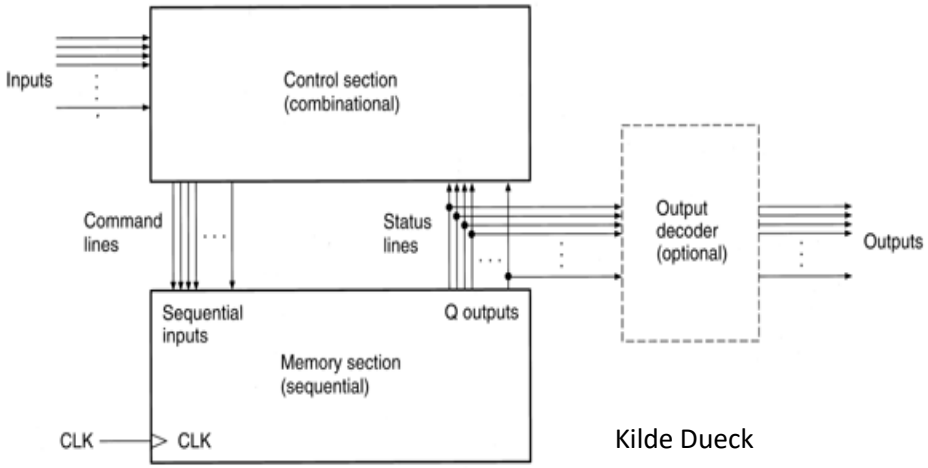


Mealy-maskin

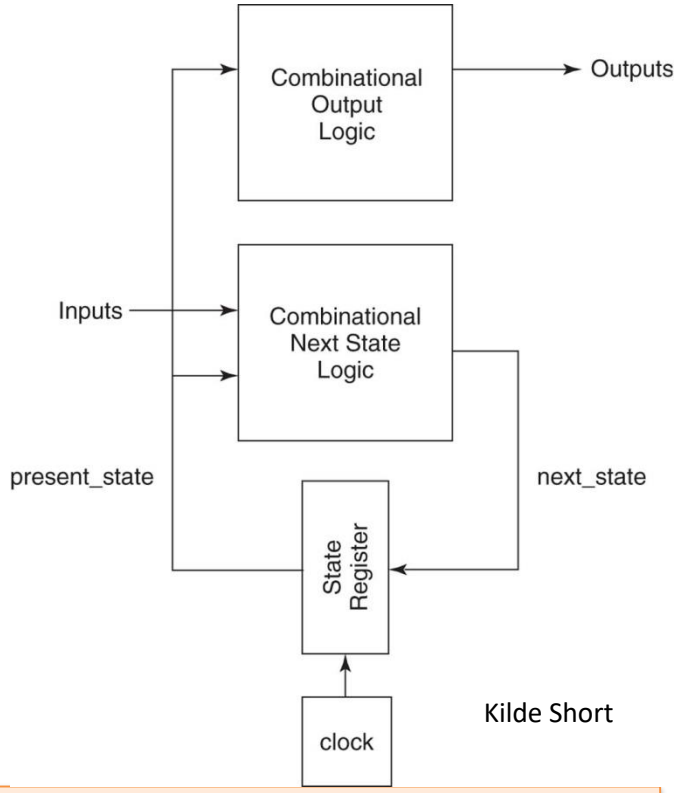


Moor-maskin

Blokkskjemaet for moor-maskinen ser slik ut



Kilde Dueck

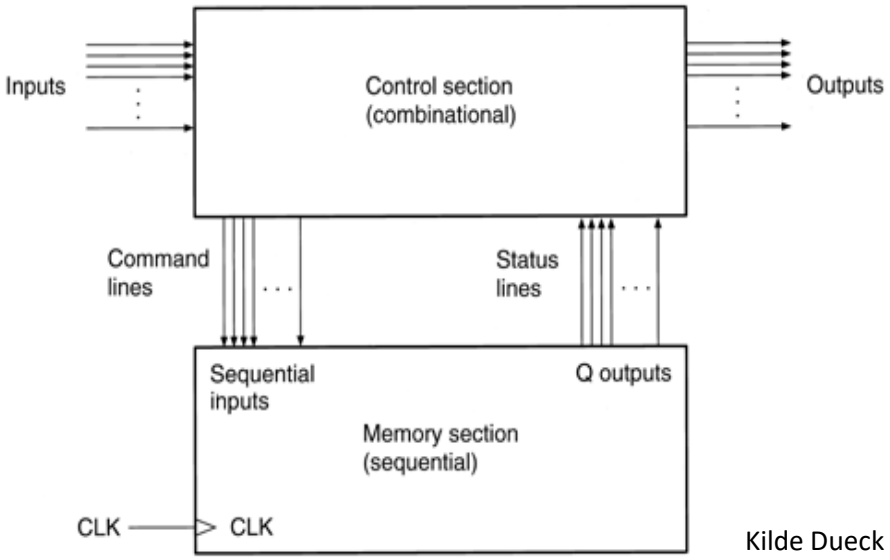


Kilde Short

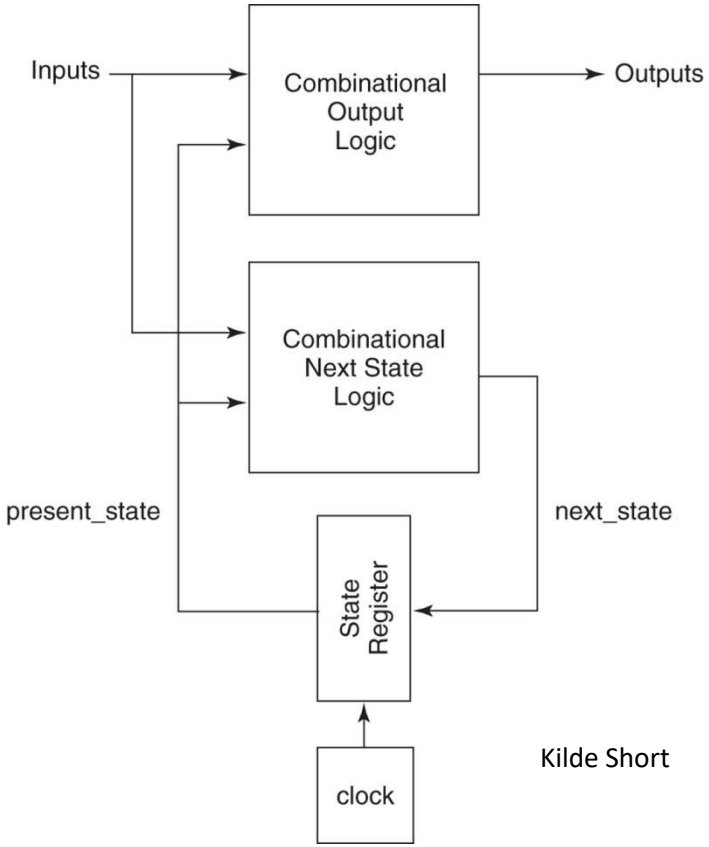
I begge disse blokkskjemaene ser vi:
Utgangene fra en Moor maskin er bare bestemt av utgangene (=NÅTILSTAND) fra registrene.
Siden utgangene, Outputs, er en kombinasjon av utsignal fra registrene, vil Outputs skifte synkront med klokken.

Mealy-maskin

Blokkskjemaet for Mealy-maskinen ser slik ut



Kilde Dueck

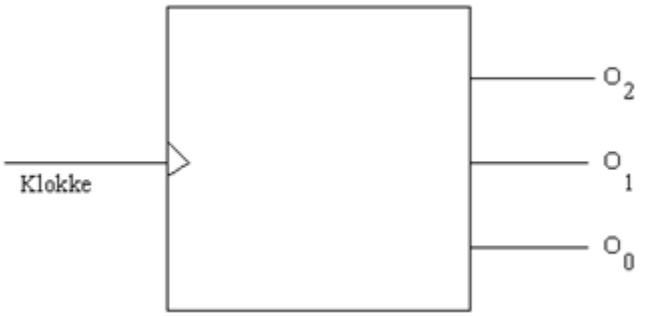


Kilde Short

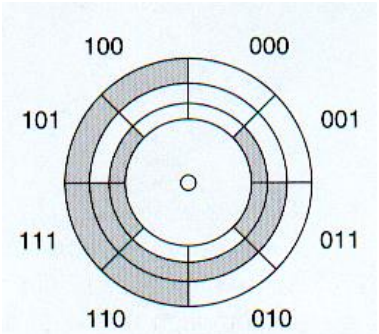
I begge disse blokkskjemaene ser vi:
Utgangene fra en Mealy maskin er bestemt både av innsignal og av utgangene (=NÅTILSTAND) for utsignal fra registrene.
Utgangene kan skifte ***asynkront*** med klokken. Dette fordi utsignalet kan også skifte dersom Inputs skifter

En enkel tilstandsmaskin med bare klokke som innsignal.

Som eksempel på det, ser vi på en tilstandsmaskin som skal produsere 3 bit Gray kode



Utgangene skal følge denne sekvensen:



8 ulike tilstander betyr av vi trenger 3 vipper. Vi ønsker dessuten å klare oss uten dekodingslogikk, og setter $O_2=Q_2$, $O_1=Q_1$ og $O_0=Q_0$.
Vi kan nå lage en tabell som viser hvordan utgangene skal endres etter hvert som sekvensen kjøres:

Table 10.1 3-bit Gray Code Sequence

$Q_2Q_1Q_0$
000
001
011
010
110
111
101
100

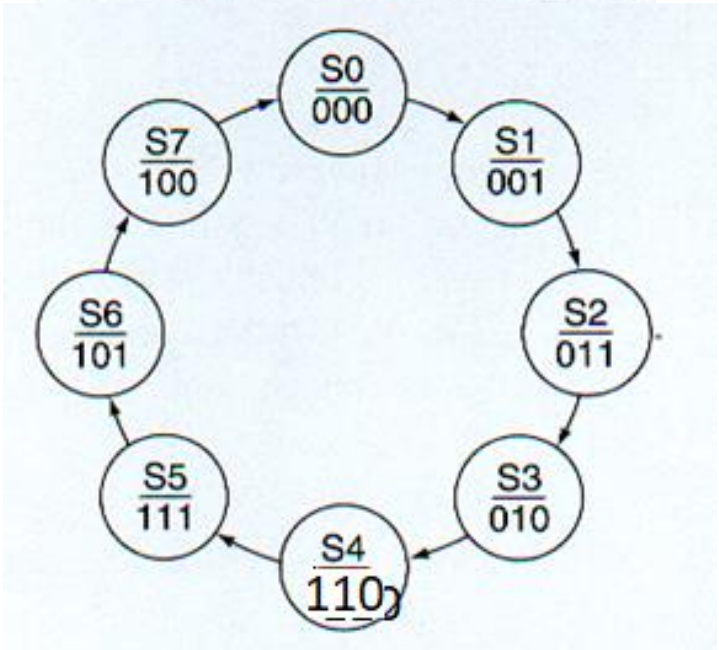
En enkel tilstandsmaskin med bare klokke som innsignal.

Ut fra tabellen lager vi så et tilstandsdiagram, og gir hver tilstand et navn, S_j, med stigende indeks.

Vi bestemmer oss for å lage kretsen med D-vipper, og lager følgende tabell:

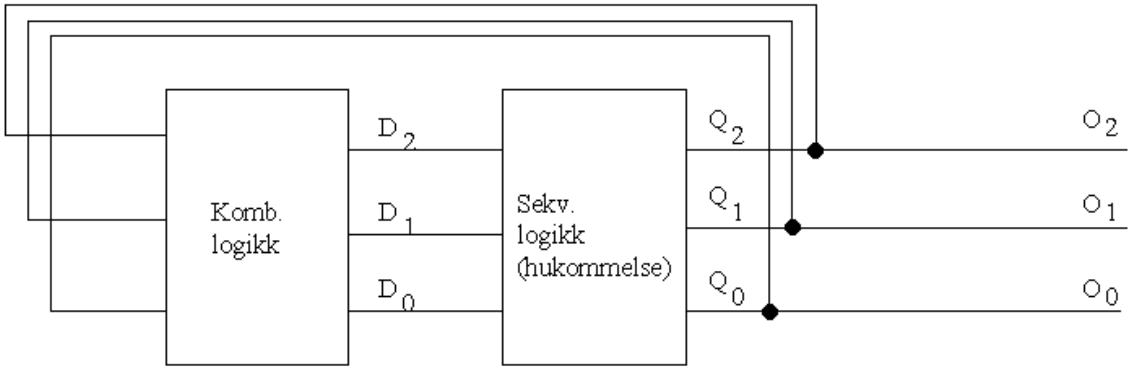
Present State	Next State	Synchronous Inputs
$Q_2Q_1Q_0$	$Q_2Q_1Q_0$	$D_2D_1D_0$
000	001	001
001	011	011
010	110	110
011	010	010
100	000	000
101	100	100
110	111	111
111	101	101

Tilstandsdiagram



En enkel tilstandsmaskin med bare klokke som innsignal.

Blokkskjema for kretsen vi se slik ut:



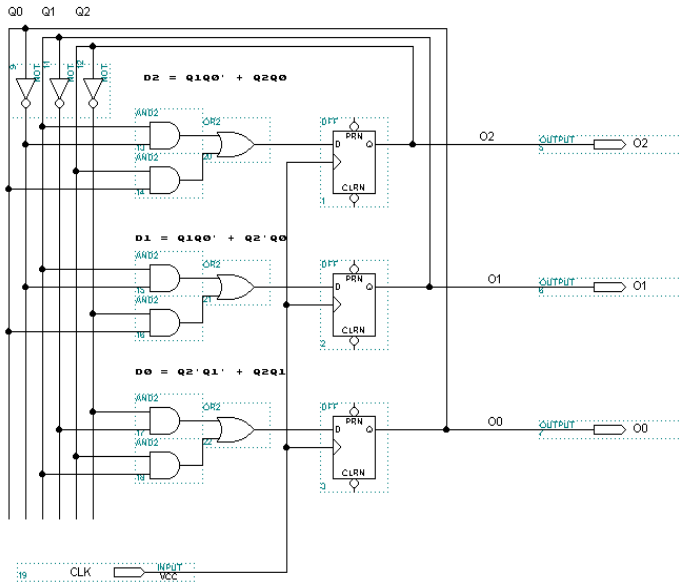
Bruker vi metodene fra digitalteknikk, finner vi ut fra tabellen, uttrykk for D_2, D_1 og D_0:

Dj = f (Q_2, Q_1, Q_0) (present state)

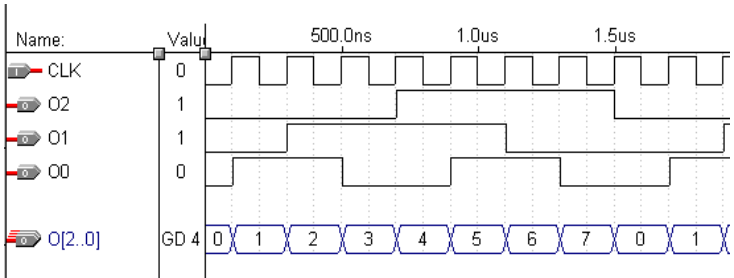
$D_0 = Q_2'Q_1' + Q_2Q_1$

$D_1 = Q_1Q_0' + Q_2'Q_0$

$D_2 = Q_1Q_0' + Q_2Q_0$



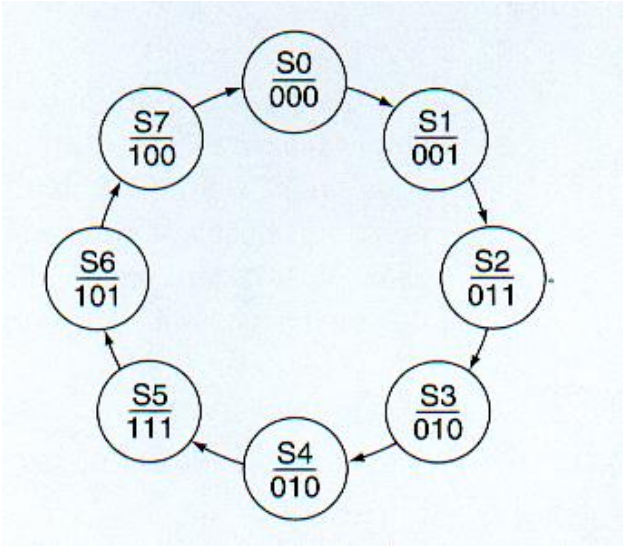
Simulering:



Så til VHDL….

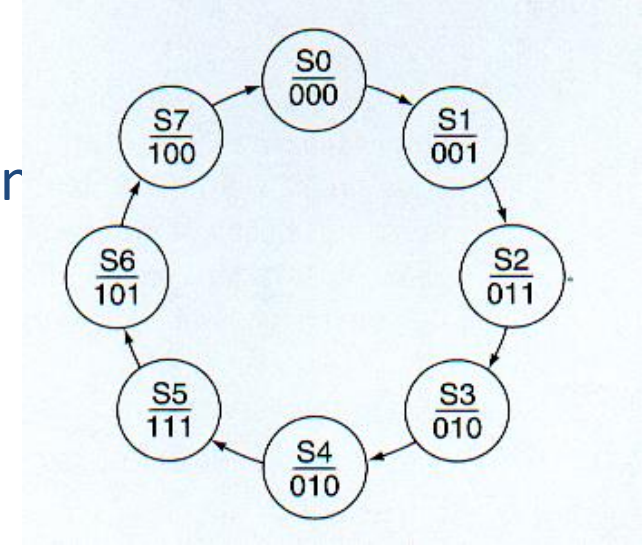
VHDL er spesielt egnet til å beskrive tilstandsmaskiner!

```
LIBRARY Ieee;
USE ieee.std_LOGIC_1164.all;

ENTITY gray_code1 is
PORT (clk : in std_logic;
        q: OUT std_logic_VECTOR(2 downto 0));
END;

ARCHITECTURE a of gray_code1 IS

TYPE state_type is (S0,s1,s2,s3,s4,s5,S6,s7);
Signal state: state_type;
BEGIN
   PROCESS(clk, reset)
   BEGIN
      if rising_edge(clk)then
         if reset = '0' then
            state <= S0;
         else
          CASE state IS
               WHEN S0 =>
                  STate <= S1;
               WHEN S1 =>
                  State <= S2;
               WHEN S2 =>
                  State <= S3;
               WHEN S3 =>
                  State <= S4;
               WHEN S4 =>
                  State <= S5;
               WHEN S5 =>
                  State <= S6;
               WHEN S6 =>
                  State <= S7;
               WHEN S7 =>
                  State <= S0;
            END CASE;
         end if;
      end if;
   END PROCESS;
WITH state SELECT
   q  <= "000" WHEN  s0,
                 "001" WHEN  s1,
                 "011" WHEN  s2,
                 "010" WHEN  s3,
                 "110" WHEN  s4,
                 "111" WHEN  s5,
                 "101" WHEN  s6,
                 "100" WHEN  s7;
END a;
```



Et tilstandsdiagram vil vanligvis inneholde nok opplysninger til at vhdl-koden kan skrives rett opp. Koden til venstre har med alt som står i tilstandsdiagrammet. Vi skal se på hvordan den er bygget opp.

Oppbygging av kode for Moor-tilstandsmaskir




```
LIBRARY Ieee;
USE ieee.std_LOGIC_1164.all;

ENTITY gray_code1 is
PORT (clk : in std_logic;
        q: OUT std_logic_VECTOR(2 downto 0));
END;

ARCHITECTURE a of gray_code1 IS

TYPE state_type is (S0,s1,s2,s3,s4,s5,S6,s7);
Signal state: state_type;

BEGIN
```

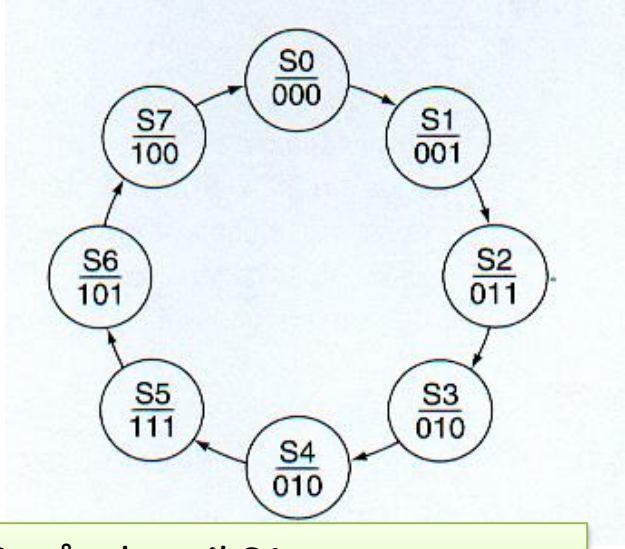
Det første vi gjør er å deklarere tilstandene.

```
TYPE state_type is (S0,s1,s2,s3,s4,s5,S6,s7);
Signal state: state_type;
```

I dette tilfeller er navnene s1,s2,s3,s4,s5,s6,s7, men det er vanlig å velge navn som beskriver hva som utføres i tilstandene

Oppbygging av kode for tilstandsmaskin

Så må vi beskrive hvordan tilstandsmaskinen (=State Maskin =SM) går fra tilstand til tilstand. Til dette benyttes CASE struktur.




```
PROCESS(clk, reset)
BEGIN
   if rising_edge(clk)then
      if reset = '0' then
         state <= S0;
      else
      CASE state IS


      END CASE;
      end if;
   end if;
END PROCESS;
```

Når SM er i tilstand S0, går den til S1 neste gang klokken trigger

```
CASE state IS
   WHEN S0 =>
      STate <= S1;

END CASE;
```

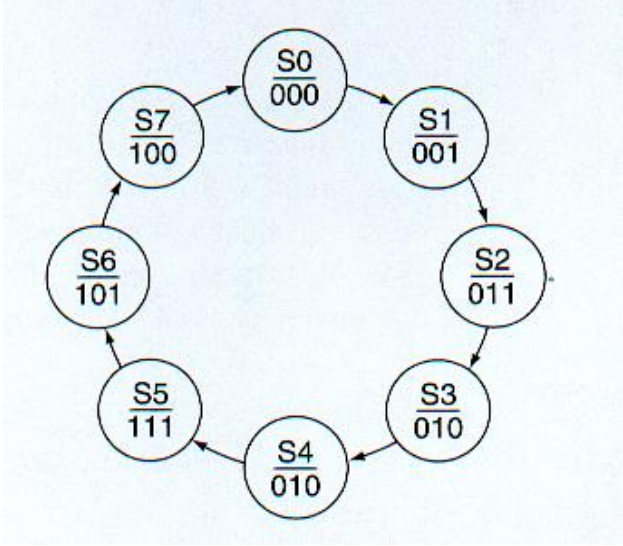
Når SM er i tilstand S1, går den til S2 neste gang klokken trigger

```
if rising_edge(clk)then
   if reset = '0' then
      state <= S0;
   else
      CASE state IS
         WHEN S0 =>
            STate <= S1;
         WHEN S1 =>
            State <= S2;

      END CASE;
   end if;
end if;
```

Oppbygging av kode for tilstandsmaskin

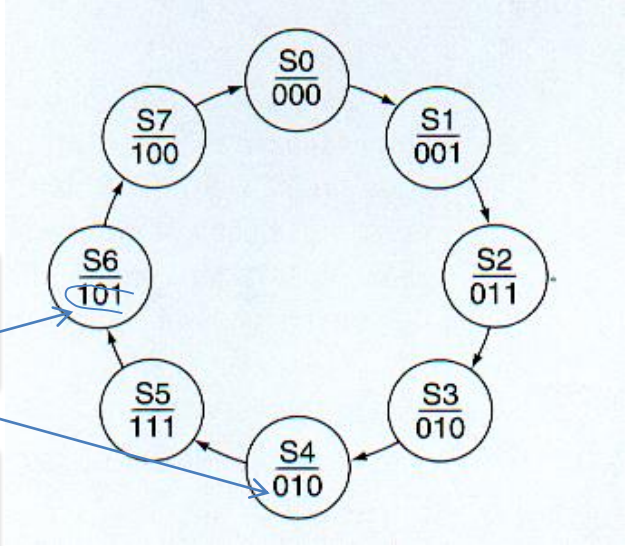
Her er SM kommet gjennom alle tilstandene

```
PROCESS(clk, reset)
BEGIN
   if rising_edge(clk)then
      if reset = '0' then
         state <= S0;
      else
       CASE state IS
            WHEN S0 =>
               STate <= S1;
            WHEN S1 =>
               State <= S2;
            WHEN S2 =>
               State <= S3;
            WHEN S3 =>
               State <= S4;
            WHEN S4 =>
               State <= S5;
            WHEN S5 =>
               State <= S6;
            WHEN S6 =>
               State <= S7;
            WHEN S7 =>
               State <= S0;
         END CASE;
      end if;
   end if;
END PROCESS;
```



Så var det utsignalene….

Oppbygging av kode for tilstandsmaskin

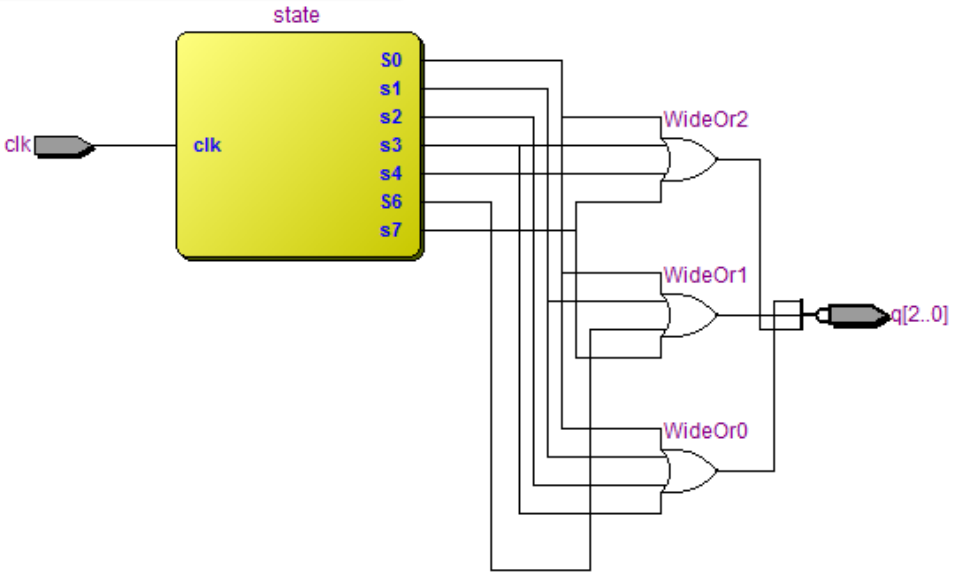


Hva utsignalene skal være er for Moormaskinen beskrevet inne i sirkelen.

Det kan gjøres på flere alternative måter
1) ved å benytte With-select-when statement.
2) Ved å benytte en prosess til
3) Ved å tilordne i case-kommandoene

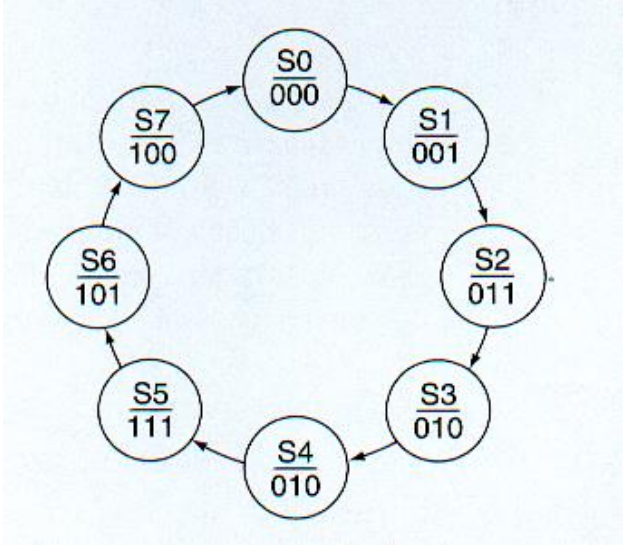
Alternativ 1: Ved å benytte With-select-when statement.

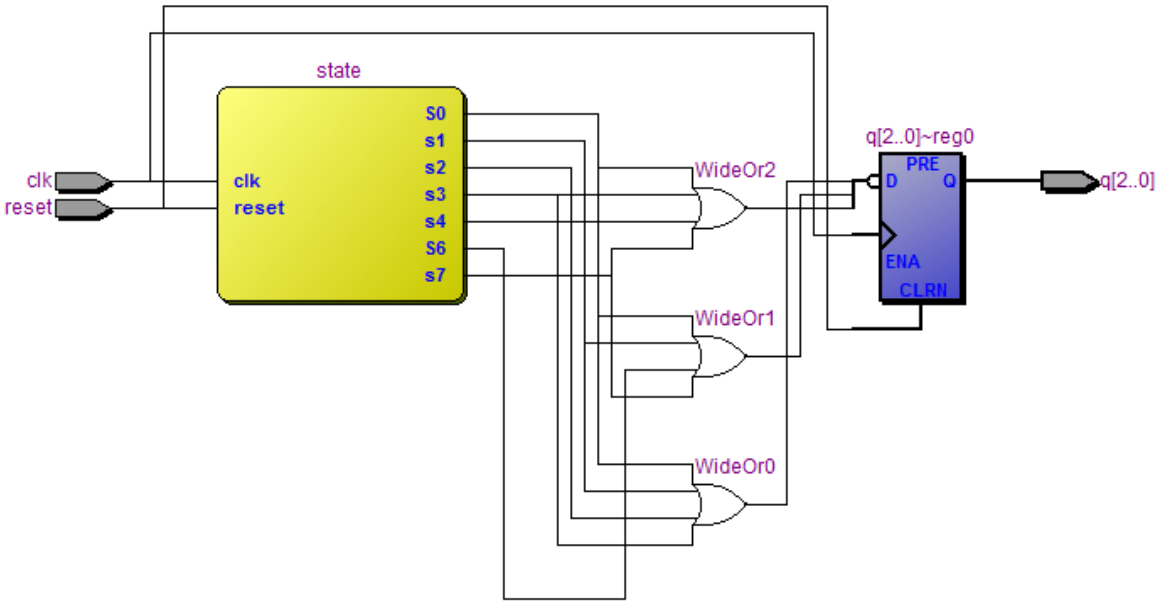
```
WITH state SELECT
   q  <= "000" WHEN  s0,
         "001" WHEN  s1,
         "011" WHEN  s2,
         "010" WHEN  s3,
         "110" WHEN  s4,
         "111" WHEN  s5,
         "101" WHEN  s6,
         "100" WHEN  s7;
END a;
```



Alternativ 2: Ved å bare ha en prosess og tilordne q verdi i case-kommandoene.

```
CASE state IS
      WHEN S0 =>
         STate <= S1;
         q <= "000";
      WHEN S1 =>
         State <= S2;
         q <= "001";
      WHEN S2 =>
         State <= S3;
         q <= "011";
      WHEN S3 =>
         State <= S4;
         q <= "010";
      WHEN S4 =>
         State <= S5;
         q <= "110";
      WHEN S5 =>
         State <= S6;
         q <= "111";
      WHEN S6 =>
         State <= S7;
         q <= "101";
      WHEN S7 =>
         State <= S0;
         q <= "100";
  END CASE;
```





Alternativ 3: Tilordning av utsignal ved å benytte en ekstra prosess:

```
PROCESS(clk, reset)
BEGIN
   if rising_edge(clk)then
      if reset = '0' then
         state <= S0;
      else
       CASE state IS
            WHEN S0 =>
               STate <= S1;
            WHEN S1 =>
               State <= S2;
            WHEN S2 =>
               State <= S3;
            WHEN S3 =>
               State <= S4;
            WHEN S4 =>
               State <= S5;
            WHEN S5 =>
               State <= S6;
            WHEN S6 =>
               State <= S7;
            WHEN S7 =>
               State <= S0;
         END CASE;
      end if;
   end if;
END PROCESS;
Process(state)
   Begin
      CASE state IS
         when S0 =>
            q <= "000";
         when S1 =>
            q <= "001";
         when S2 =>
            q <= "011";
         when S3 =>
            q <="010";
         when S4 =>
            q <="110";
         when S5 =>
            q <="111";
         when S6 =>
            q <="101";
         when S7 =>
            q <="100";
       End CASE;
   End process;
END a;
```