INF3340

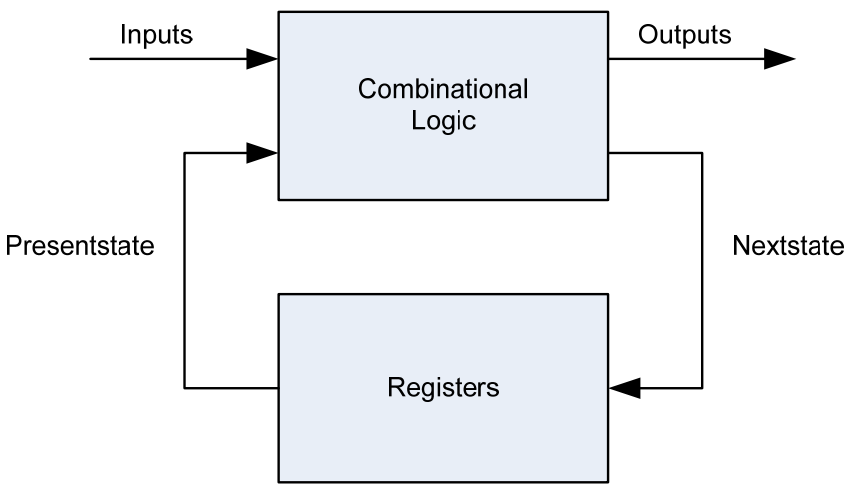
Tilstandsmaskiner

Innhold

- Tilstandsmaskiner
- Mealy og Moore maskiner
- ASM tilstandsdiagrammer
- Syntese av ASM diagrammer
- Tilstandskoding
- Implementasjon ved bruk av VHDL
- Eksempler

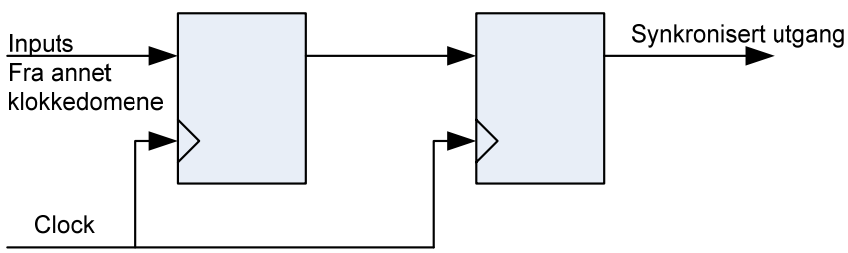
Sekvensielle systemer

- Utganger avhenger av tidligere og nåværende verdier på innganger
- Sekvensielle systemer er synkrone. Dvs. styrt/synkronisert av en klokke.
  - Registre benyttes som lager
  - Tidligere innganger benyttes til å styre systemet inn i en tilstand (Presentstate)
- Modelleres ofte som i figuren nedenfor:
  - Nåværende tilstand (Presentstate) er lagret i registre.
  - Innganger kombineres med nåværende tilstand for å danne neste tilstand (Nextstate) og nye utgangsverdier



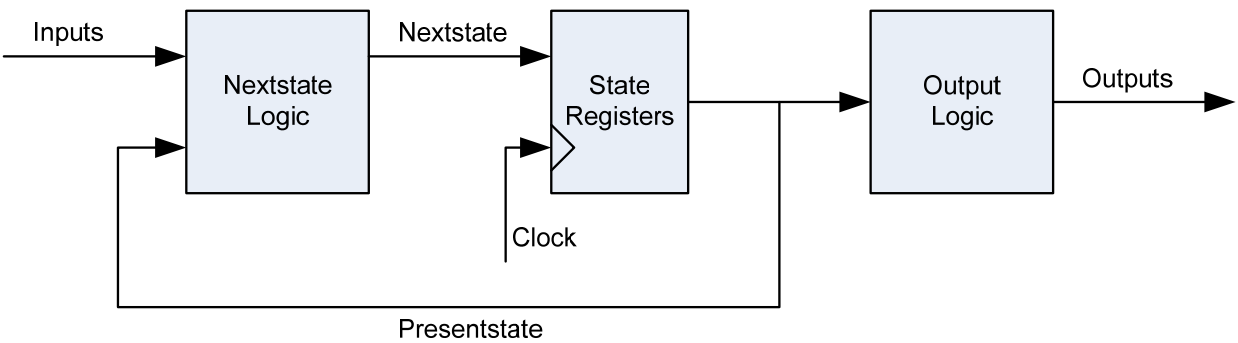
Moore og Mealy maskiner

- Synkrone sekvensielle systemer modelleres oftest som en Mealy eller Moore maskin.
- Mealy og Moore kalles for tilstandsmaskiner
- Nåværende tilstand er lagret i det vi kaller tilstandsregistre (Stateregisters)
- NB! Pass på at setup og holdetider for tilstandsregistre er overholdt
  - Dersom ikke er det fare for metastabilitet.
- Dersom innganger kommer fra annet klokkedomene har man ikke kontroll på om setup og holdetider er overholdt.
  - Må synkronisere inngangene.



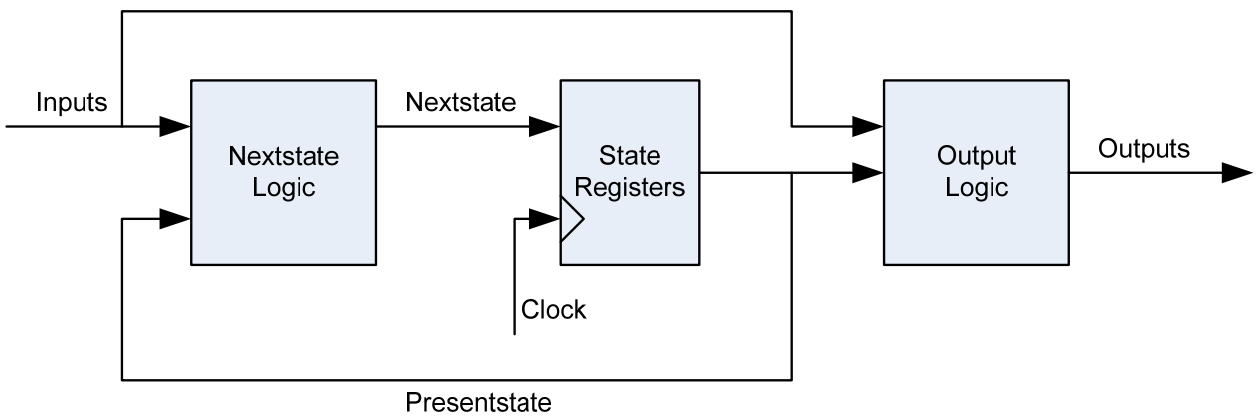
Moore maskin

- I en Moore tilstandsmaskin er utgangene en ren funksjon av nåværende tilstand (Presentstate)
- I en Moore maskin er det en klokkeperiode forsinkelse fra inngang til utgang



Mealy maskin

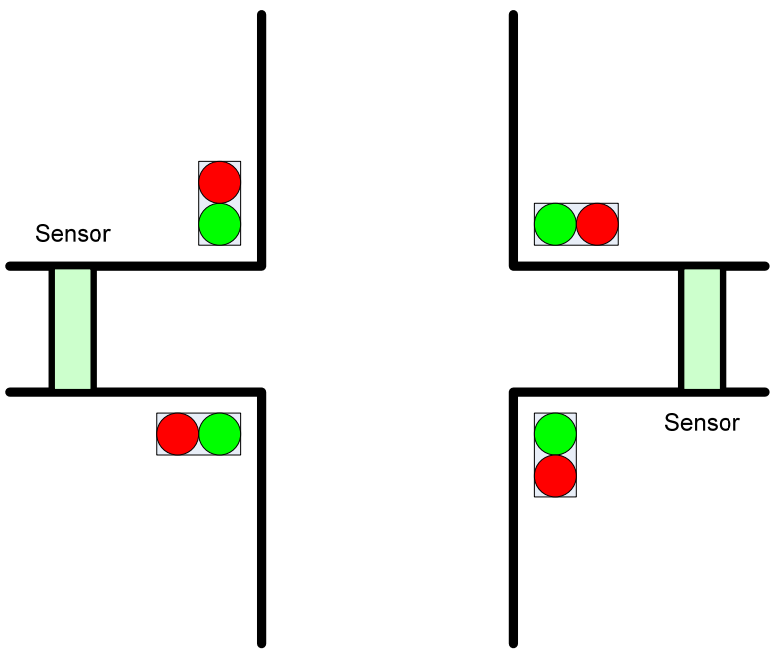
- I en Mealy tilstandsmaskin er utgangene en funksjon av nåværende tilstand og inngangene.
- Raskere
  - Mindre forsinkelse fra inngang til utgang
- Mer komplisert dekoding av utganger



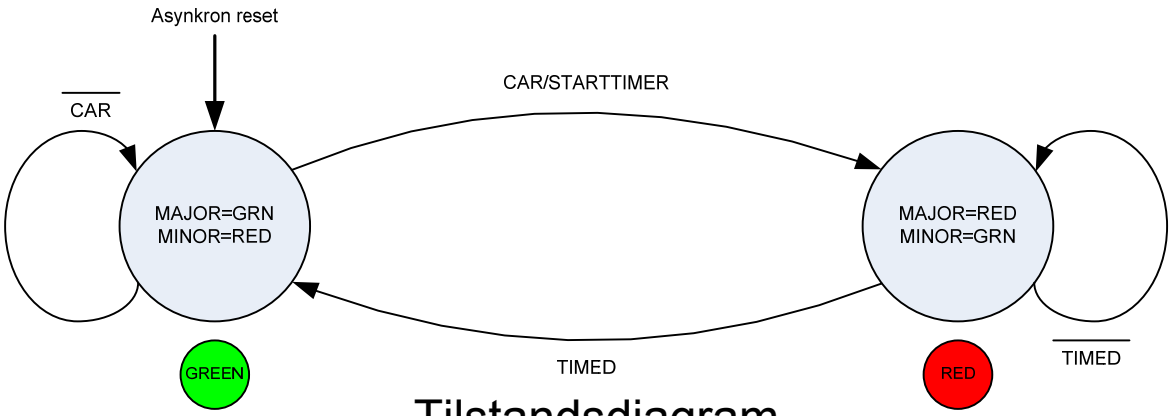
Tilstandsdiagrammer

- Eksempel: Enkel trafikklysstyring mellom en hovedvei og to mindre sideveier

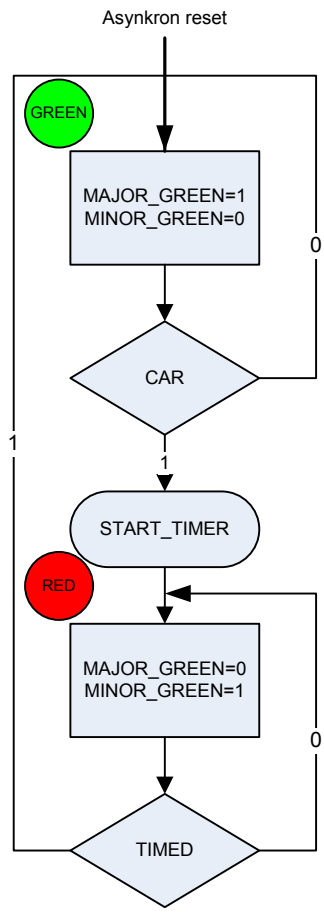
- Virkemåte:
  - Sensorer detekterer at det kommer en bil på en sidevei.
  - Gir sideveien grønt lys/hovedvei rødt
  - Sideveien har grønt lys i en viss tid gitt av en timer



Beskrivelse av tilstandsmaskiner



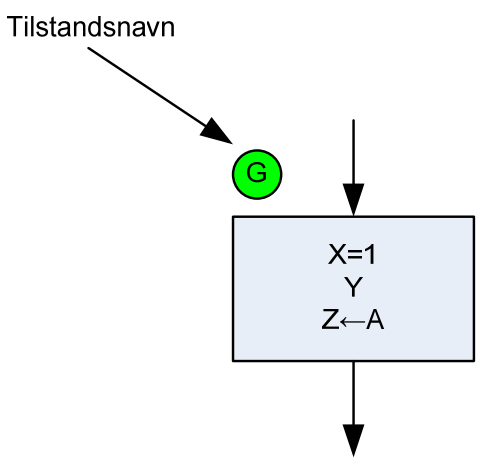
Tilstandsdiagram



ASM (Algorithmic State Machine) diagram

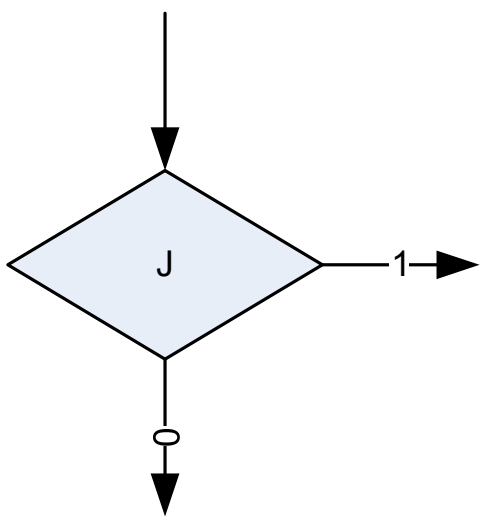
ASM diagrammer

- Tilstandsboks
  - Tar en klokkeperiode å gjennomføre
  - X=1
    - Tilordning i hver tilstand
  - Y
    - Satt til 1 i denne tilstand og 0 ellers
  - Z←A
    - Blir tilordnet verdien A i skiftet fra en tilstand til en annen. Verdien blir lagret til den eventuelt skifter verdien i et annet tilstandsskifte (register lagring)



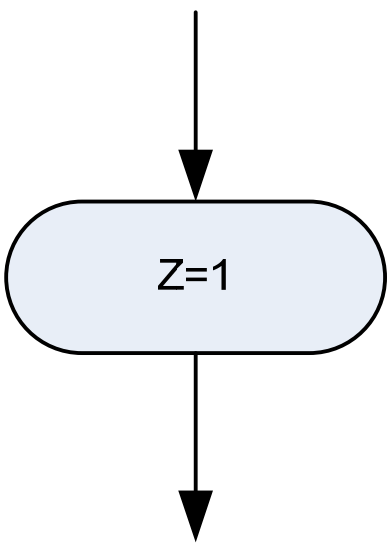
ASM diagrammer

- Avgjørelsesboks (Decisionbox)
  - To eller flere avgreininger basert på verdien på en eller flere innganger
  - Avgjørelsesboksen må følge en tilstandsboks og være assosiert med en tilstand
  - Derfor vil avgjørelser bli tatt i samme periode som tilstanden varer
    - Innganger må være stabile ved starten av klokkeflanken (setup og holdetider)

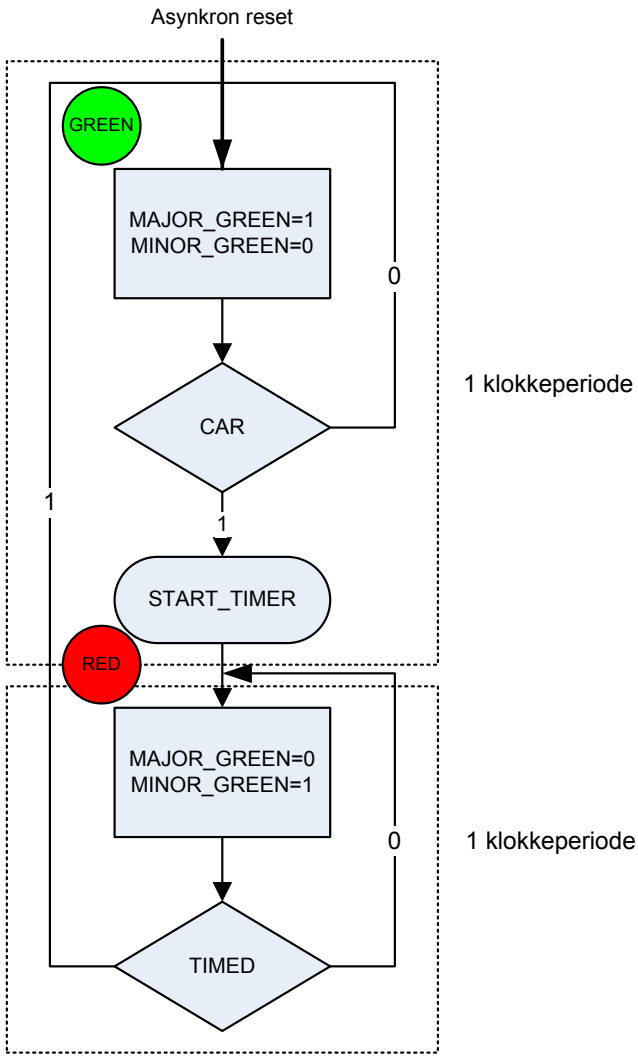


ASM diagrammer

- Betingede utgangsverdier
  - Må følge en avgjørelsesboks
  - Utgang kan skifte verdi i løpet av tilstanden
  - Mealy utgang

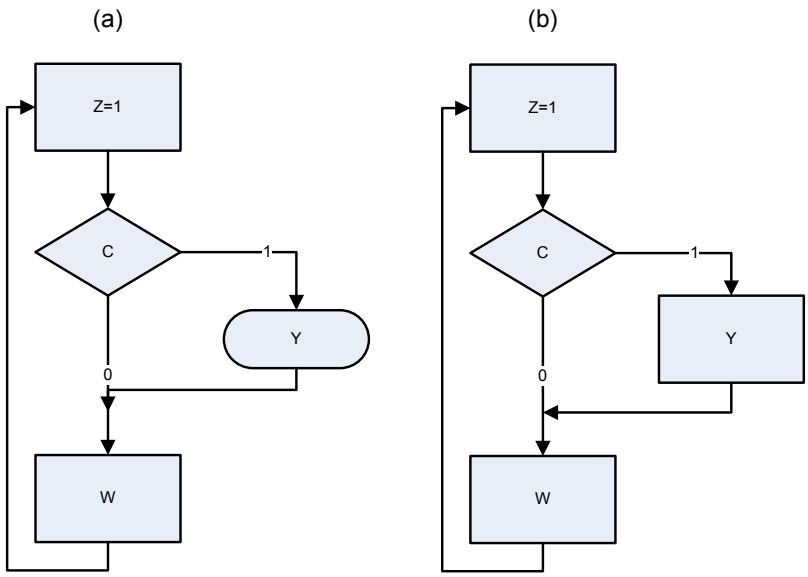


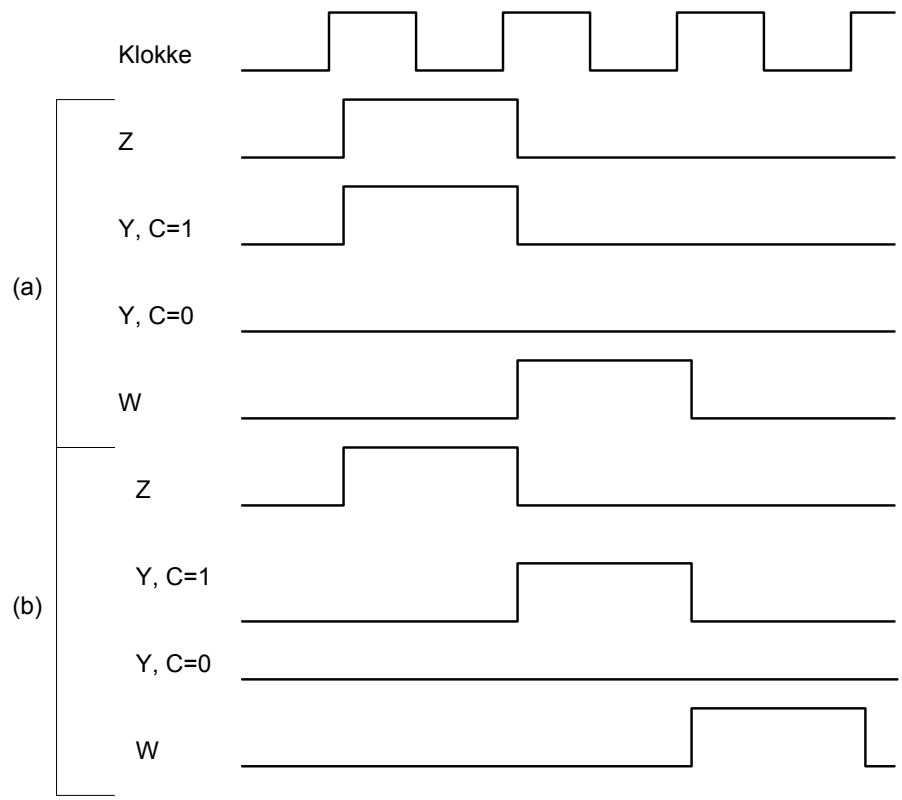
ASM diagrammer



- En tilstand varer fra en tilstandsboks til neste.
- Dette tilsvarer en klokkeperiode

ASM diagrammer





Syntese av ASM diagrammer

- Klassisk metode for å implementere en tilstandsmaskin er å sette opp
  - Sannhetstabeller for neste tilstand
  - Sannhetstabell for utganger
  - Bruke Karnaughdiagrammer for å finne (minimerte) logiske ligninger
- Dette krever at man tilordner tilstandsregistrene bestemte verdier for hver tilstand.
  - Ønsker å tilordne verdier på tilstander slik at nestetilstandslogikk og utgangslogikk blir enklest mulig, men dette blir fort svært vanskelig fordi det blir tidlig svært mange forskjellige kombinasjoner
  - Hvis vi vil tilordne s tilstander med en minimum antall D-flip-flop'er (bit) m, må $2^{m-1} < s \leq 2^m$
  - Da er antall forskjellige kombinasjoner av tilstandskodinger gitt ved

$$n := \frac{\left(2^m\right)!}{\left(2^m - s\right)!}$$

  - s=3, m=2, gir n=24
  - s=5, m=3, gir n=6720
  - s=9, m=4, gir n=4151347200

Syntese av ASM diagrammer

- Eksempel på tilordning av tilstandsvektorer der s=3 (A,B,C), m=2

00	A	A	A	A	A	A	B	B	B	B	B	B	C	C	C	C	C	C						
01	B		B	C		C	A		A	C		C	A		A	B		B	A	A	B	B	C	C
10	C	B		B	C		C	A		A	C		B	A		A	B		B	C	A	C	A	B
11		C	C		B	B		C	C		A	C		B	B		A	A	C	B	C	A	B	A

- Ingen kjent metode for å avgjøre hvem tilordning som er best

Vanlige kodinger

- Bruk asynkron reset ved oppstart
  - Starttilstandsvektoren kan da kodes med 0..00
- Normal binær tellesekvens kan benyttes for etterfølgende tilstander
  - Vanligst koding i CPLD'er
- Koding etter Graysekvens gir minimale skifter mellom tilstander
- Bruk av et bit pr. tilstand gir enklest mulig neste tilstandslogikk.
  - Populær koding i FPGA pga. mange flip-flop'er og "smal" logikk (4-input LUT i hvert nivå).
  - Kalles "one-hot" koding

Implementasjon ved bruk av VHDL

- Vanligst å kode tilstandsvektoren ved å bruke enumererte datatyper
  - Definisjon
    - type state_type is (GREEN,RED);
    - signal presentstate,nextstate : state_type;
  - Fordel:
    - Tilstandsvektorkoding gjøres i syntesen
    - Enumererte datatyper gir meget god observerbarhet i RTL-simuleringer
- Tilstandsmaskin deles opp i to/tre prosesser:
  - Tilordning presentsstate <= nextstate i en klokket prosess
  - Kombinatorisk prosess for å finne nextstate (og utganger)
    - case for teste nåværende tilstand
    - if for finne neste tilstand
  - (Eventuell egen prosess for å definere utganger)

Trafikkontroller

- Enitity, definisjon av datatype og stateregister

```
 1: library IEEE;
 2: use IEEE.std_logic_1164.all;
 3:
 4: entity TRAFFIC is
 5:   port
 6:   (
 7:     CLOCK          : in std_logic;
 8:     RESET          : in std_logic;
 9:     TIMED          : in std_logic;
10:     CAR            : in std_logic;
11:     START_TIMER    : out std_logic;
12:     MAJOR_GREEN    : out std_logic;
13:     MINOR_GREEN    : out std_logic
14:   );
15: end entity TRAFFIC;
16:
17: architecture RTL_TRAFFIC of TRAFFIC is
18:
19:   type state_type is (GREEN, RED);
20:   signal PRESENT_STATE, NEXT_STATE : state_type;
21:
```

Trafikkontroller

- Stateregister

```
22: begin
23:    --Stateregister
24:    STATE_REG:
25:    process (CLOCK,RESET) is
26:    begin
27:      if RESET = '1' then
28:        PRESENT_STATE <= Green;
29:      elsif (rising_edge(CLOCK)) then
30:        PRESENT_STATE <= NEXT_STATE;
31:      end if;
32:    end process STATE_REG;
```

Traffikkontroller

- Nestetilstands og utgangslogikk

```
34:   --Nextstate logic and output logic
35:   COMB:
36:   process (CAR, TIMED, PRESENT_STATE) is
37:   begin
38:     START_TIMER <= '0';
39:     case PRESENT_STATE is
40:       when GREEN =>
41:         MAJOR_GREEN <= '1';
42:         MINOR_GREEN <= '0';
43:         if (CAR = '1') then
44:           START_TIMER <= '1';
45:           NEXT_STATE <= RED;
46:         else
47:           NEXT_STATE <= GREEN;
48:         end if;
49:       when RED =>
50:         MAJOR_GREEN <= '0';
51:         MINOR_GREEN <= '1';
52:         if (TIMED = '1') then
53:           NEXT_STATE <= GREEN;
54:         else
55:           NEXT_STATE <= RED;
56:         end if;
57:     end case;
58:   end process COMB;
```

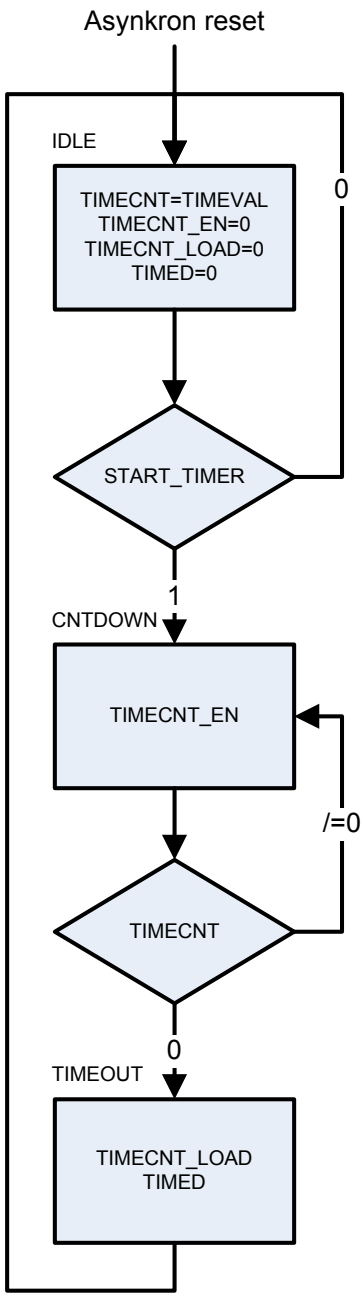
Traffikkontroller

- Timed telleren er en tilstandsmaskin i seg selv og kan implementeres som en teller eller en tilstandsmaskin
- Dette er eksempel på to tilstandsmaskiner som henger sammen (linked)

```
 85:   TIMER:
 86:   process (RESET,CLOCK)
 87:   begin
 88:     if RESET = '1' then
 89:       TIMED <= '0';
 90:       TIMERCNT <= others => '0';
 91:     elsif rising_edge(CLOCK) then
 92:       TIMED    <= '0';
 93:       if START_TIMER = '1' then
 94:         TIMERCNT <= TIMER_VALUE;
 95:       elsif timercnt > 0 then
 96:         TIMERCNT <= TIMERCNT - 1;
 97:         if TIMERCNT = 1 then
 98:           TIMED <= '1';
 99:         end if;
100:       end if;
101:     end if;
102:   end process;
```

Timer

- Moore maskin for timer



Timer-entity

```
 1: library IEEE;
 2: use IEEE.std_logic_1164.all;
 3: use IEEE.numeric_std.all;
 4:
 5: entity TIMER is
 6:   generic
 7:   (
 8:     N : natural := 16;
 9:     TIMERCONST : natural :=  65535
10:   );
11:   port
12:   (
13:     CLOCK          : in std_logic;
14:     RESET          : in std_logic;
15:     START_TIMER    : in std_logic;
16:     TIMED          : out std_logic
17:   );
18: end entity TIMER;
```

Timer-stateregister

```
22:    type TIMER_STATE is (IDLE,CNTDOWN,TIMEOUT);
23:    signal TIMER_ST, NEXT_TIMER_ST : TIMER_STATE;
24:    signal TIMERCNT_LOAD  : std_logic;
25:    signal TIMERCNT_EN    : std_logic;
26:    signal TIMERCNT : unsigned(N-1 downto 0);
27:
28: begin
29:    --Stateregister
30:    STATE_REG:
31:    process (RESET,CLOCK) is
32:    begin
33:      if RESET = '1' then
34:        TIMER_ST <= IDLE;
35:      elsif (rising_edge(CLOCK)) then
36:        TIMER_ST <= NEXT_TIMER_ST;
37:      end if;
38:    end process STATE REG;
```

Timer-nextstate

```
40:   --Nextstate logic and output logic
41:   COMB:
42:   process (START_TIMER, TIMER_ST,TIMERCNT) is
43:   begin
44:     --Default verdier
45:     TIMED          <= '0';
46:     TIMERCNT_EN    <= '0';
47:     TIMERCNT_LOAD <= '0';
48:     NEXT_TIMER_ST <= IDLE;
49:     case TIMER_ST is
50:       when IDLE =>
51:         if START_TIMER = '1' then
52:           NEXT_TIMER_ST <= CNTDOWN;
53:         else
54:           NEXT_TIMER_ST <= IDLE;
55:         end if;
56:       when CNTDOWN =>
57:         TIMERCNT_EN <= '1';
58:         if TIMERCNT = 0 then
59:           NEXT_TIMER_ST <= TIMEOUT;
60:         else
61:           NEXT_TIMER_ST <= CNTDOWN;
62:         end if;
63:       when TIMEOUT =>
64:         TIMED <= '1';
65:         TIMERCNT_LOAD <= '1';
66:         NEXT_TIMER_ST <= IDLE;
67:     end case;
68:   end process;
```

Timer-counter

```
70:   TIMECOUNTER:
71:   process(RESET,CLOCK)
72:   begin
73:     if RESET = '1' then
74:       TIMERCNT <= to_unsigned(TIMERCONST,N);
75:     elsif rising_edge(clock) then
76:       if TIMERCNT_LOAD = '1' then
77:         TIMERCNT <= to_unsigned(TIMERCONST,N);
78:       elsif TIMERCNT_EN = '1' then
79:         TIMERCNT <= TIMERCNT-1;
80:       end if;
81:     end if;
82:   end process;
```