Forelesning 07
Begrenset Tilstandsmaskin
Finite State Machine

Hieu Nguyen

Innledning

- Finite State Machine (FSM) blir brukt å modellere et system som transitterer mellom et begrenset antall tilstander
- Overganger avhenger av **gjeldende/nåværende** maskintilstand og **eksterne** inngangssignaler
- Ulikhet med regelmessige sekvensielle kretser, har FSM **ikke** enkelt og gjentatt mønster i sin tilstand-overgang
- Den next-state-logic modulen må oppbygges fra **bunn**, kaller vi den «**tilfeldig** logikk/ random logic»

Synkron Design - FSM

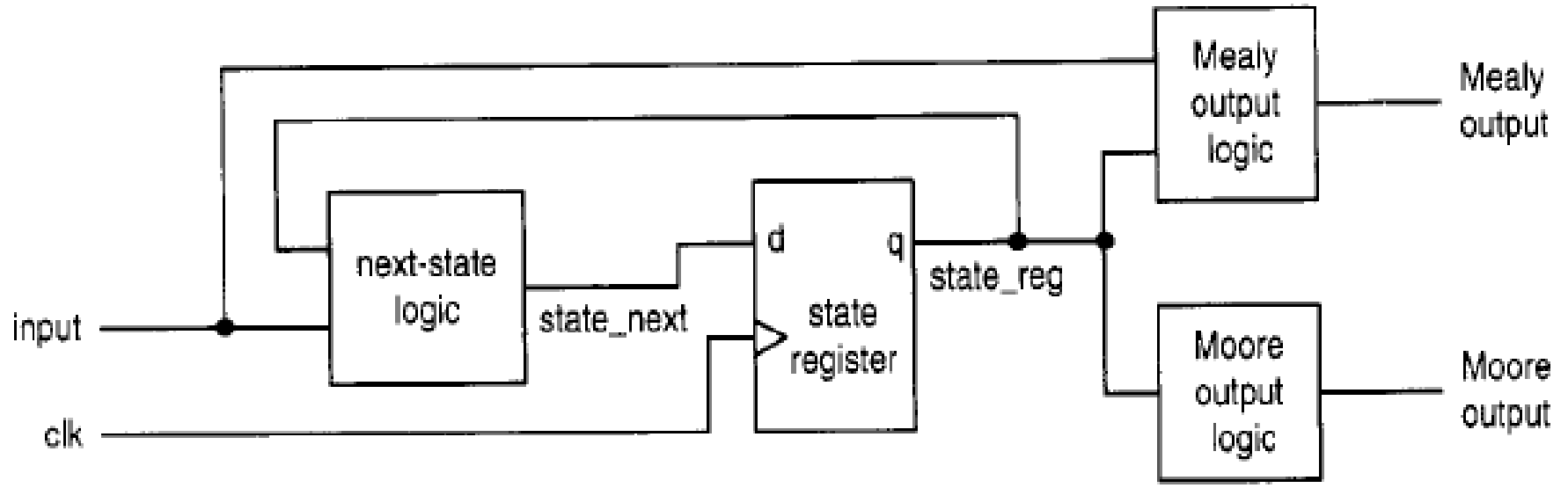


Figure 5.1 Block diagram of a synchronous FSM.

Synkron Design - FSM

Next-state-logic: kombinasjonskrets

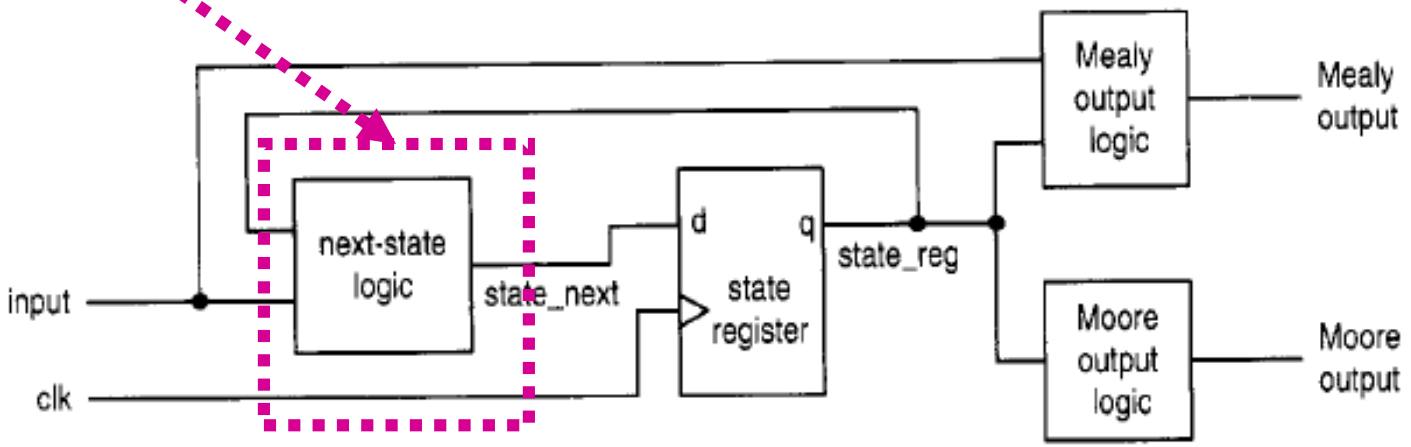


Figure 5.1 Block diagram of a synchronous FSM.

Synkron Design - FSM

State-register: sekvensiell krets

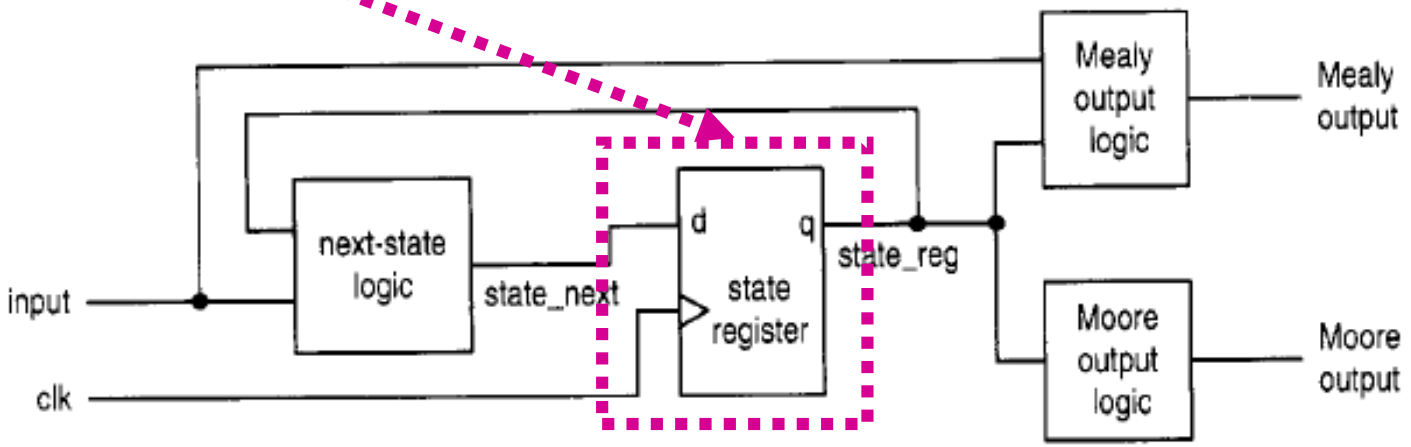


Figure 5.1 Block diagram of a synchronous FSM.

Synkron Design - FSM

Output logic: kombinasjonskrets

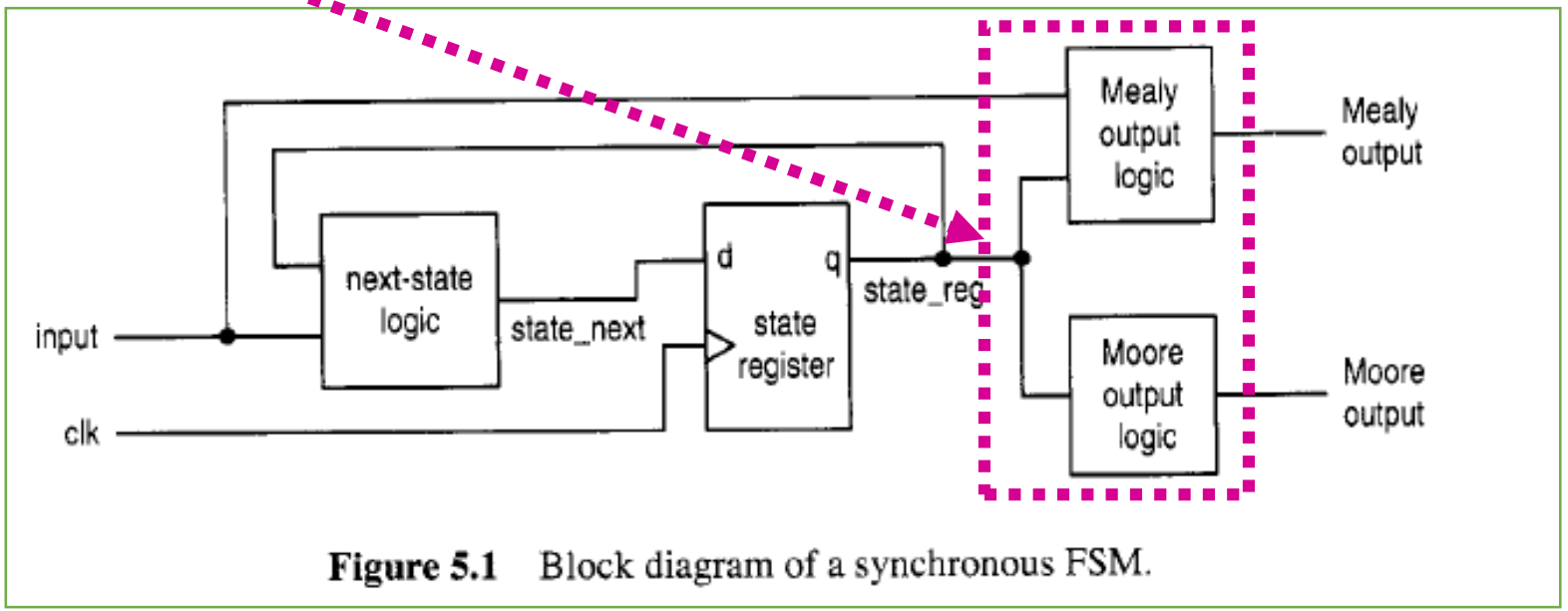


Figure 5.1 Block diagram of a synchronous FSM.

Synkron Design - FSM

Moore FSM: Utgangssignaler er avhengige av kun state-register

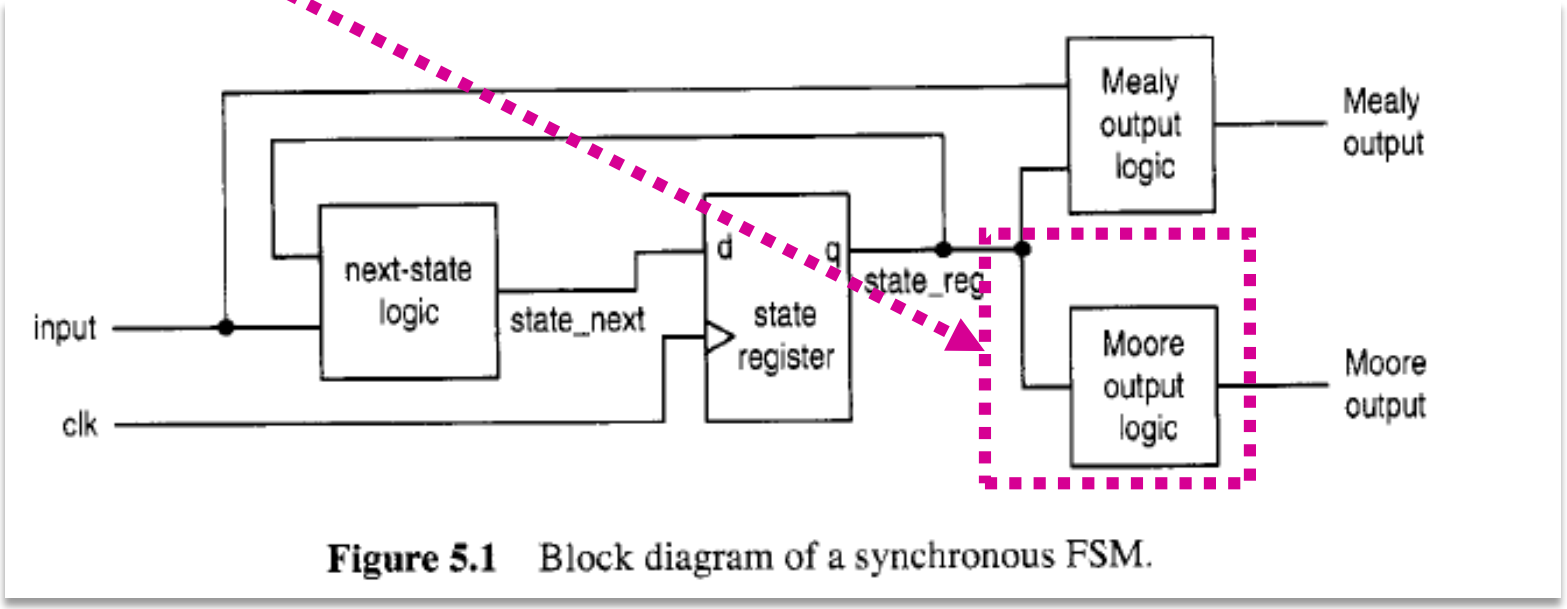


Figure 5.1 Block diagram of a synchronous FSM.

Synkron Design - FSM

Mealy FSM: Utgangssignaler er avhengige av både state-register og eksterne signaler

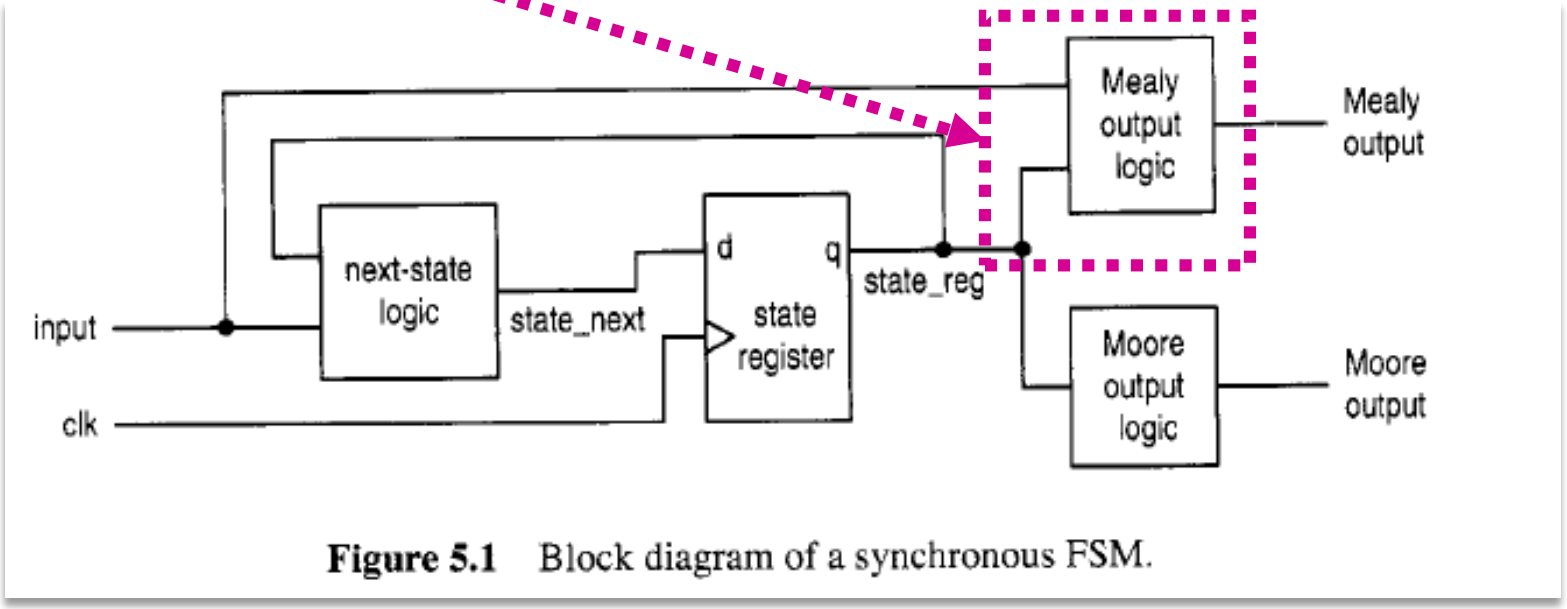
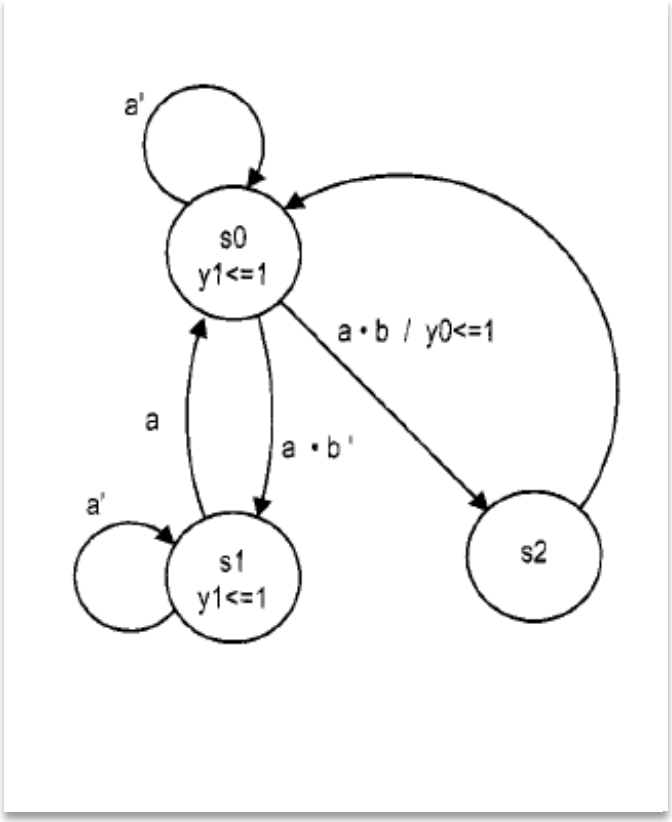


Figure 5.1 Block diagram of a synchronous FSM.

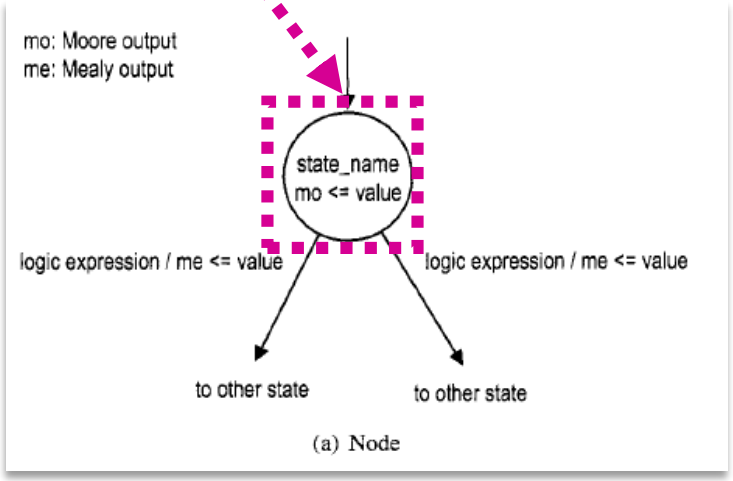
FSM Representasjon
Abstract State Diagram

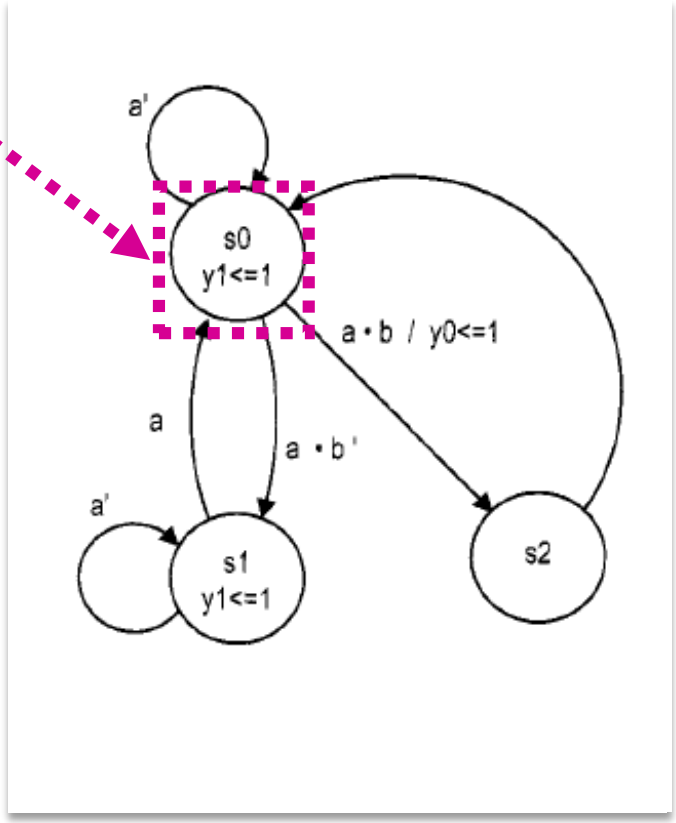
- **Node**: representerer tilstander og blir tegnet med sirkler
- **Overgangs**-sirkelbuer/arcs
- **Logiske uttrykk**
  - Representere en spesifikk **betingelse** for overgang
  - Overgang blir utført når logisk uttrykk assosiert med den blir evaluert «**sant/true**»



FSM Representasjon
Abstract State Diagram

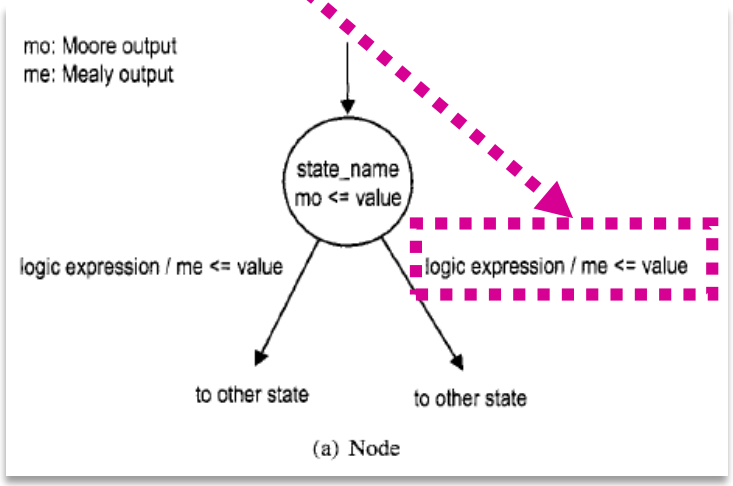
Moore-utgangsverdier blir plassert inn i sirklene fordi de er avhengige av gjeldende tilstander

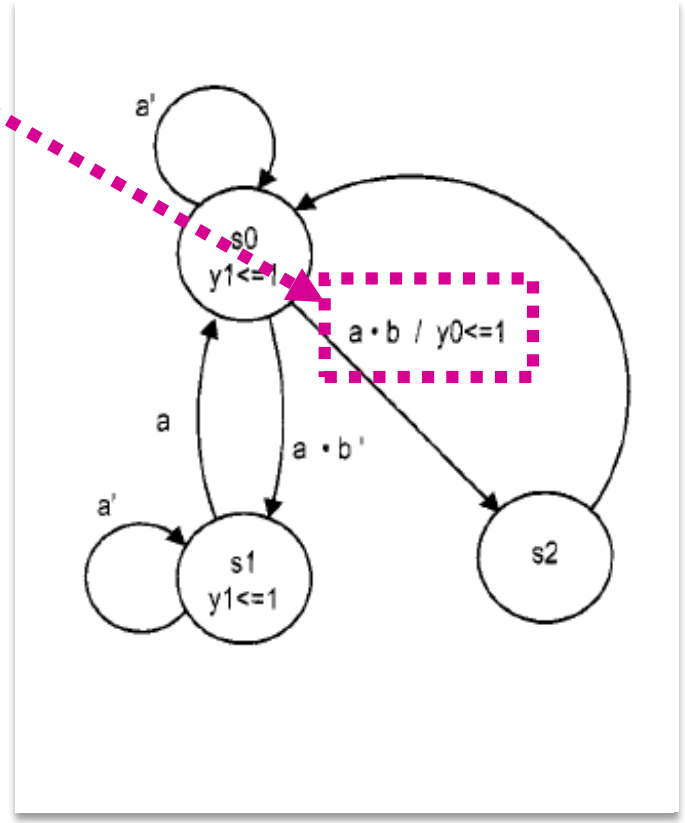




FSM Representasjon
Abstract State Diagram

Mealy-utgangsverdier blir assosiert med betingelser av overgang siden de er avhengige av gjeldene tilstand og eksterne signaler





Utvikling av VHDL-Koder

- Rutinen å utvikle VHDL-koder for FSM er **samme** for regelmessige kretser
  - Vi skiller **state-register** fra **next-state** logic og **output logic**
  - Hovedforskjellen mellom FSM og Regelmessig krets er **next-state-logic**
  - For FSM, følger next-state-logic flyten av tilstandsdiagram eller ASM-flytskjemaet

Utvikling av VHDL-Koder
FSM-modellering

- For klarhet og fleksibilitet, bruker vi VHDL-nummererert datatype til å representere FSM. **Malen er**

```
type  NAME_FSM  is (tilstand_1, tilstand_2,.., tilstand_n);
```

- For eksempel: Det tidligere tilstandsdiagrammet har tre tilstander: **s0, s1, s2**, og vi bruker bruker-definert nummerert datatype slik

```
type FSM_Eksempel is (S0, S1, S2);
```

- Vi kan definere signaler med bruker-definert datatype slik

```
signal state_reg, state_next : FSM_Eksempel;
```

Utvikling av VHDL-Koder
FSM-modellering

- Architecture

```
-------------Architecture-------------------
architecture architecture_name of entity_name is

   -------- FSM-deklarasjon her-------------

   type FSM_name is (tilstand_1, tilstand_2, ..., tilstand_n);

   signal state_reg, state_next : FSM_name;
   ----------------------------------------

   ---------- Others internal signals deklaration-
   -- Add your code here
   ------------------------------------------------
   begin

      ------------ state-register part------------
         --Add your code here
      ------------------------------------------

      ------------------ next state logic part---
      -- Add your code here
      ------------------------------------------

      --------------Output logic-----------------
      -- Add your code here
      ------------------------------------------
   end architecture_name;
```

FSM modellering: mellom architecture og begin

Utvikling av VHDL-Koder
FSM-modellering

- Architecture

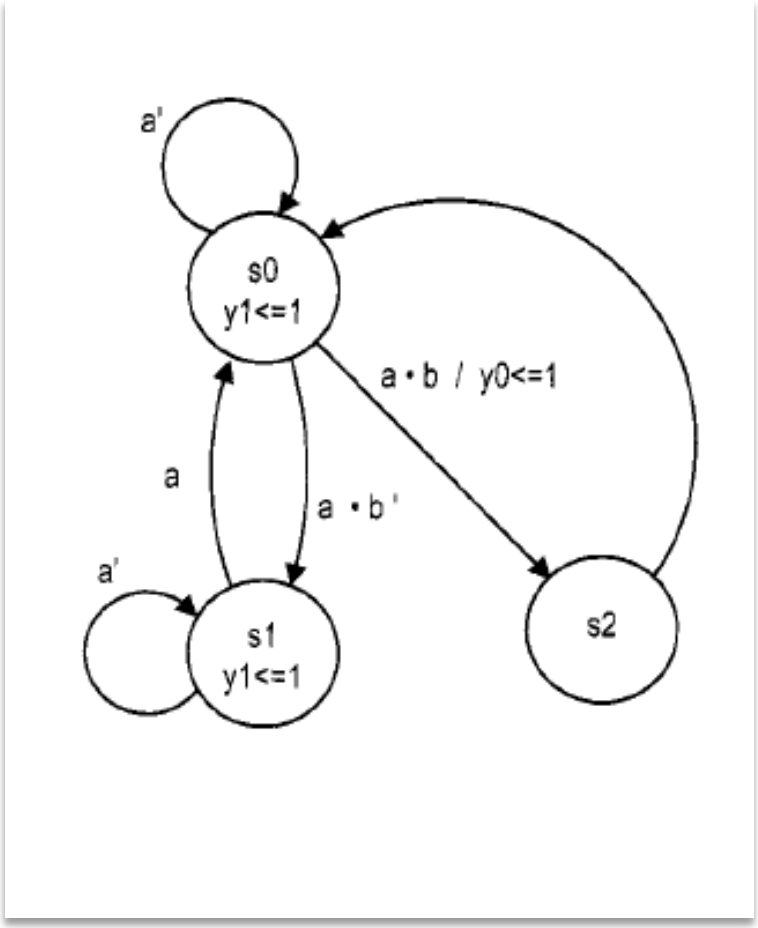
```
--------------Architecture--------------------
architecture architecture_name of entity_name is

   -------- FSM-deklarasjon her-------------

   type FSM_name is (tilstand_1, tilstand_2, ..., tilstand_n);

   signal state_reg, state_next : FSM_name;
   ----------------------------------------

   ---------- Others internal signals deklaration-
   -- Add your code here
   ------------------------------------------------
   begin

      ------------ state-register part------------
         --Add your code here
      ---------------------------------------------

      ------------------ next state logic part---
      -- Add your code here
      ------------------------------------------

      --------------Output logic-----------------
      -- Add your code here
      ------------------------------------------

   end architecture_name;
```

State_register del: sekvensiell modellering

Next_state_logic: kombinasjonal krets

Output logic: kombinasjonal krets

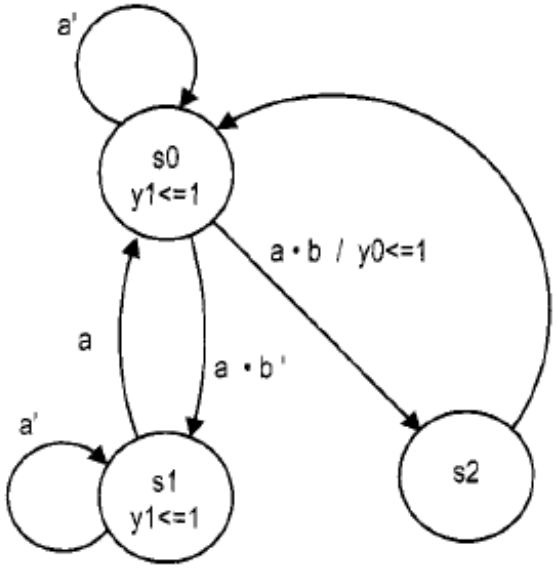
Utvikling av VHDL-Koder Eksempel



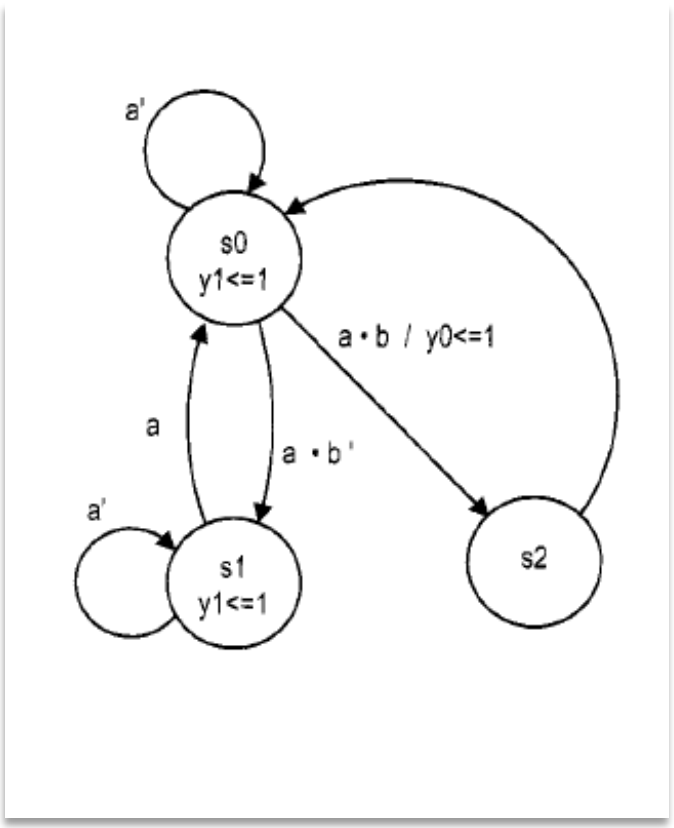
Utvikling av VHDL-Koder
Eksempel

State register: malen

```
process (clk, reset)
   begin

      if(reset = '1') then

         state_reg  <= initial_state;

      elsif (clk'event and clk = '1') then

         state_reg  <= state_next;

      end if;
   end process;
```



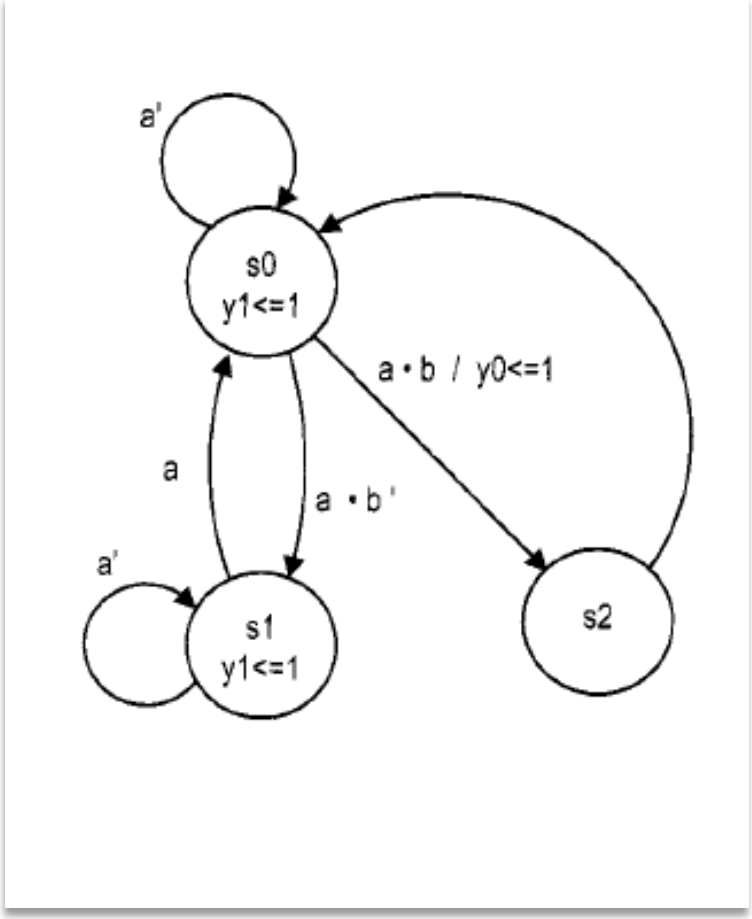
Utvikling av VHDL-Koder
Eksempel



Next state-template

```
process(sensitivity_list)

begin
   case state_reg is
      when s0 =>
         ---------add your codes here-----

      when s1 =>
         ---------add your codes here-----
      when s2 =>
         ---------add your codes here-----
      when others =>
         ---------add your codes here-----
   end case;

end process;
```

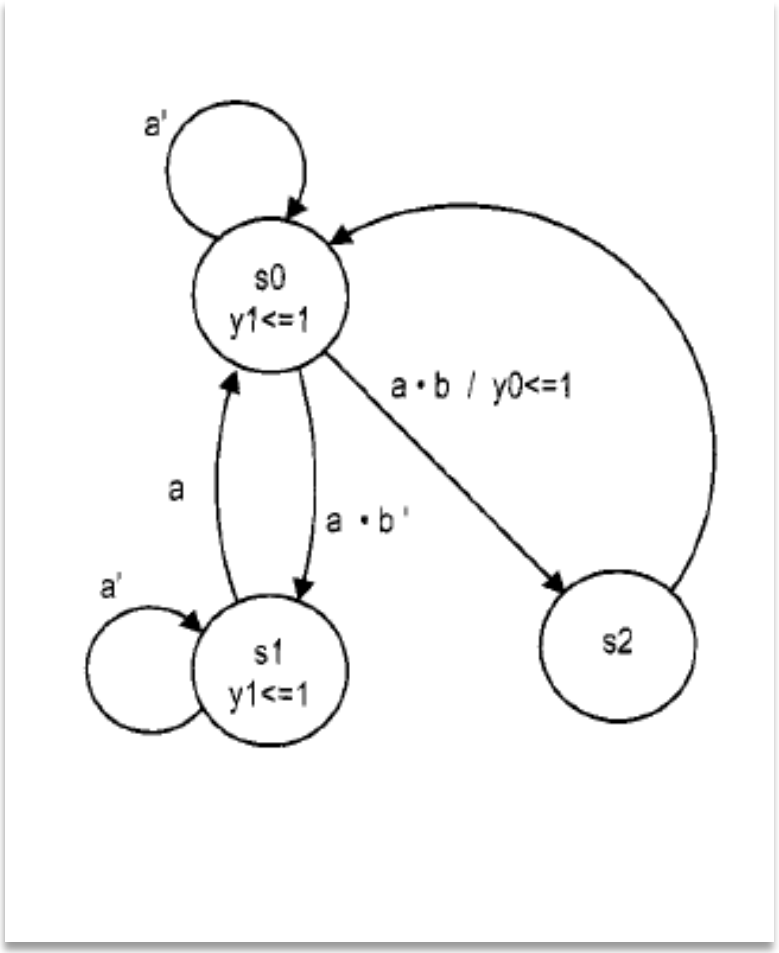
Utvikling av VHDL-Koder
Eksempel



Moore output logic: Malen

```
process(state_reg)
begin
   case state_reg is

      when state_1 =>
         ----- Add your code here---

      when sate_2 =>
         ----- Add your code here---
      when state_n =>
         ----- Add your code here---
   end case;
end process;
```

Utvikling av VHDL-Koder
Eksempel



Mealy output logic: Malen

```
process(state_reg, input_signal_1, input_signal_2,.., input_signal_n)
begin
   case state_reg is

      when state_1 =>
         ----- Add your code here---

      when sate_2 =>
         ----- Add your code here---
      when state_n =>
         ----- Add your code here---
   end case;
end process;
```

Homework & Exam Preparation

- Edge detection
- Debouncing circuit