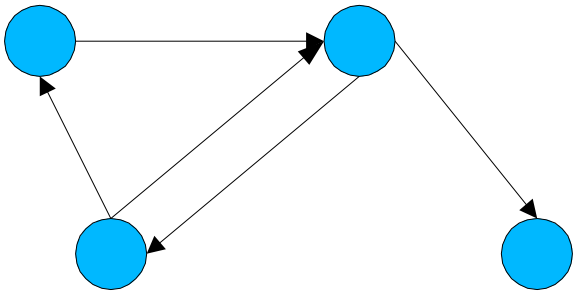
Modellierung von FSMs



Finite State Machines

- Endliche Automaten werden zur Implementierung von Steuerwerken verwendet
  - Steuerwerke können sehr komplex sein
- Ein Automat besitzt Ein– und Ausgangssignale sowie eine Zustandsmenge (sequentielles Verhalten)

Typen von Automaten

- Moore–Automaten: Der Wert der Ausgangssignale hängt nur vom aktuellen Zustand ab
  - $A = (I, O, S, \delta, \lambda)$, $\delta : I \times S \rightarrow S$, $\lambda : S \rightarrow S$
- Mealy–Automaten: Der Wert der Ausgangssignale hängt auch von den Eingangssignalen ab
  - $A = (I, O, S, \delta, \lambda)$, $\delta : I \times S \rightarrow S$, $\lambda : I \times S \rightarrow S$

Kodierung der Zustände

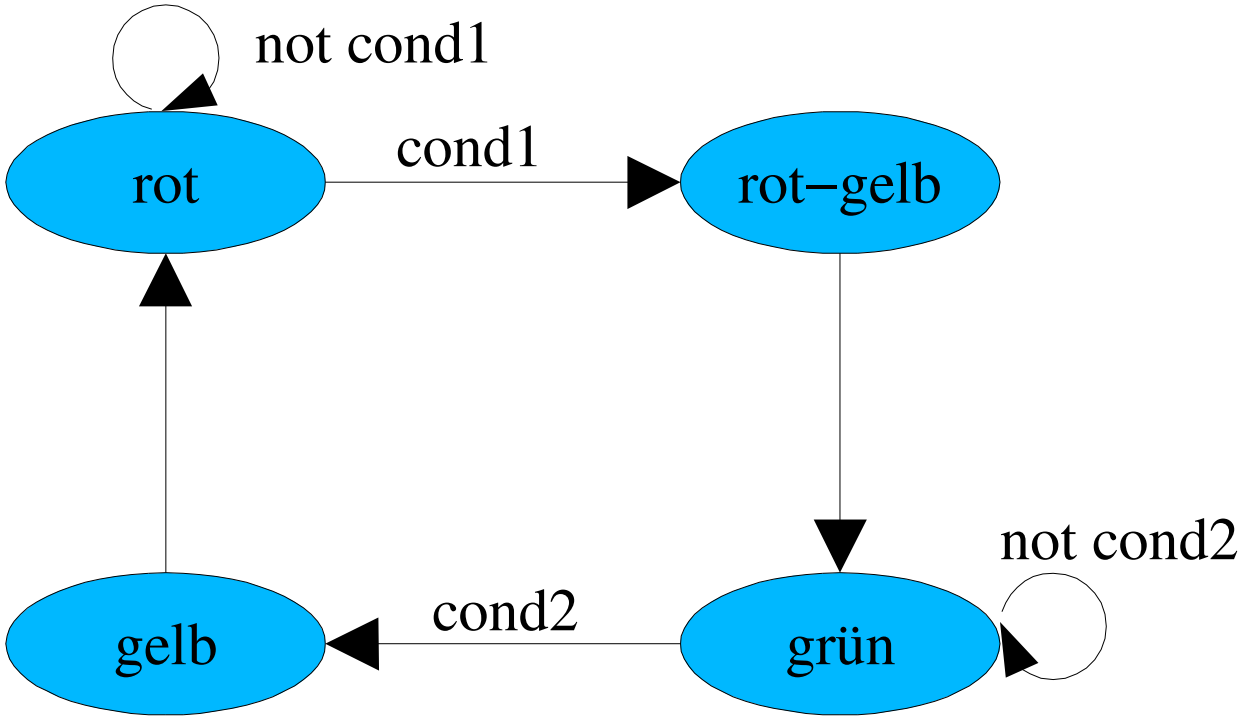
- Zur Kodierung von n Zuständen sind mindestens $\lceil \log_2(n) \rceil$ Bits notwendig
- Die Kodierung kann einen Einfluss haben auf
  - die Größe der Schaltung
  - die Verzögerungszeit / den Stromverbrauch
- Spezielle Kodierungen können von Vorteil sein
  - One Hot – es ist immer genau ein Bit auf ’1’ gesetzt

Beschreibung der Zustände

- Beschreibung mit einem Enum–Typ
  - ```
    type Ampel_Zustand is
                    ( rot, rotgelb, gelb, gruen );
    ```
  - Das Synthese–Tool kann (muss) die Kodierung festlegen
- Explizite Festlegung
  - ```
    type Ampel_Zustand is
                      std_logic_vector(2 downto 0);
    constant rot      : Ampel_Zustand := "100";
    constant rotgelb  : Ampel_Zustand := "110";
    ```
  - Der Designer legt die Kodierung fest

Modellierung des Automaten

- State Transition Graph des Moore–Automaten



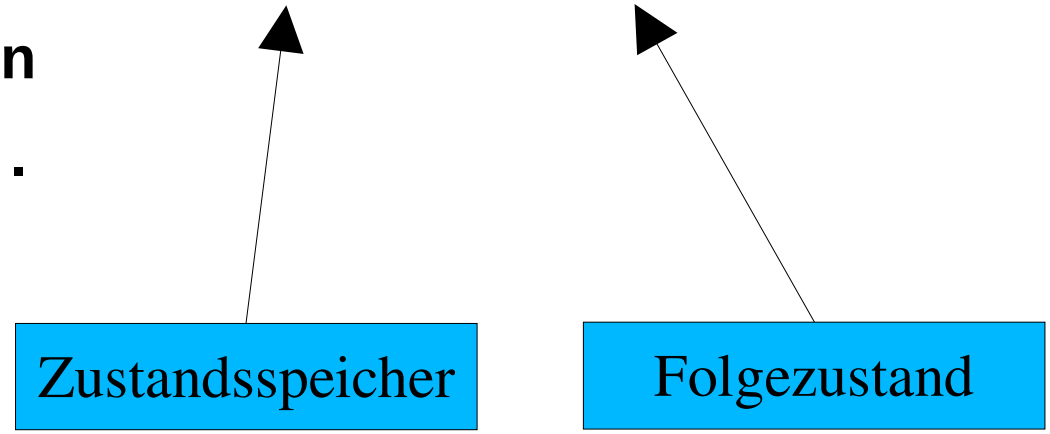
VHDL: Entity

```
library ieee;
use ieee.std_logic_1164.all;

entity ampel_steuerung is
   port(clk, reset   : in std_logic;
        cond1, cond2 : in std_logic;
        o_rot, o_gelb, o_gruen
                     : out std_logic);
end ampel_steuerung;
```

VHDL: Architecture

```
architecture rtl of ampel_steuerung is
  type state_type is
                ( rot, rotgelb, gelb, gruen );
  signal state, next_state : state_type;
begin
  ...
```



Output Logic

Keine Abhängigkeit von Inputs (Moore–Automat)

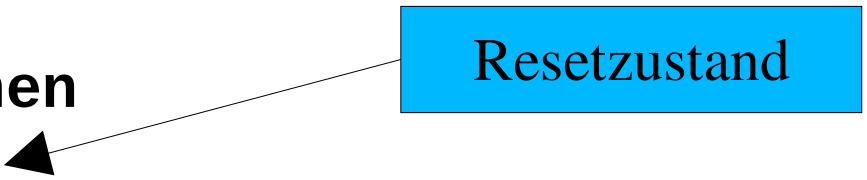
```
...
begin
   output_logic : process (state)
   begin
      case state is
      when rot =>
  o_rot <= '1'; o_gelb <= '0'; o_gruen <= '0';
      when rotgelb =>
  o_rot <= '1'; o_gelb <= '1'; o_gruen <= '0';
      when gelb =>
  o_rot <= '0'; o_gelb <= '1'; o_gruen <= '0';
      when gruen =>
  o_rot <= '0'; o_gelb <= '0'; o_gruen <= '1';
      end case;
   end process;
   ...
```

State Logic

```
...
state_logic : process (state, cond1, cond2)
begin
   case state is
   when rot =>
          if cond1 = '1'
          then next_state <= rotgelb;
          else next_state <= rot;
          end if;
   when rotgelb =>
          next_state <= gruen;
   when gruen =>
          if cond2 = '1' ...
   when gelb =>
          next_state <= rot;
   end case;
end process;
...
```

Folgezustand setzen

```
    ...
    state_reg : process (clk, reset)
    begin
       if reset = '1' then
          state <= rot;
       elsif clk'event and clk = '1' then
          state <= next_state;
       end if;
    end process;
end rtl;
```



Synthese

- FSMs sollten immer in drei Prozessen kodiert werden
  - Übersichtlichkeit / Wartbarkeit
- Wo möglich sollte die Kodierung explizit angegeben werden
  - Grund: Verifikation
- Wenn Enums verwendet werden, so sollte der Reset–Zustand der linke Enum–Wert sein