Steuerpfad

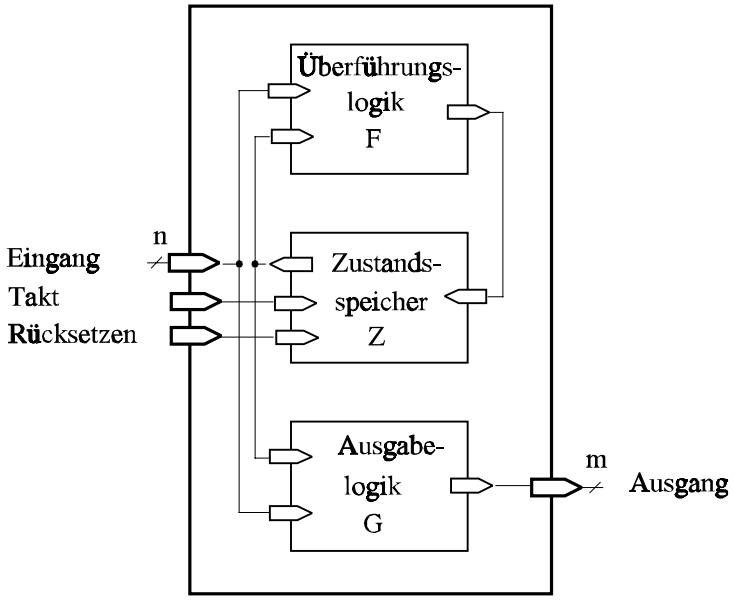
Der Steuerpfad dient zur Erzeugung von Steuersignalen. Die erzeugten Steuersignale hängen vom Bearbeitungsstand ("Zustand") der Aufgabe und von Eingangsgrößen des Steuerpfades (z.B. auszuführende Funktion, Operation, Status, Sensorsignale) ab. Die Steuersignale können unter anderem auf Datenpfade, andere Steuerpfade oder Aktoren einwirken. Steuerpfade können als endliche bzw. erweiterte endliche Automaten modelliert werden.

Ein endlicher Automat kann durch ein Quintupel A = (X,Y,Z,f,g) beschrieben werden, in dem **X** die Menge der **Eingangsgrößen** (alle zugelassenen Wertebelegungen des Eingangsvektors), **Z** die Menge der **inneren Zustände** (alle zugelassenen Wertebelegungen des Zustandsvektors) und **Y** die Menge der **Ausgangsgrößen** (alle zugelassenen Wertebelegungen des Ausgangsvektors) darstellen.

Die **Überführungsfunktion f**: $X \times Z \rightarrow Z$ ist eine eindeutige Abbildung des Kreuzproduktes X x Z in Z, die **Ausgabefunktion g**: $X \times Z \rightarrow Y$ eine eindeutige Abbildung von X x Z in Y.

Die Arbeitsweise eines **synchronen** (taktgesteuerten) **Automaten A** kann wie folgt charakterisiert werden: Ist $z^t \in Z$ der Zustand (Wert des Zustandsvektors) von A im Takt t und sind $x^t \in X$ die Werte der in diesem Takt anliegenden Eingangsgrößen (Wert des Eingangsvektors), so erzeugt A im Takt t die Werte der Ausgangsgrößen $y^t = g(x^t,z^t)$ (Wert des Ausgangsvektors), und A nimmt im Takt t+1 den Zustand $z^{t+1} = f(x^t,z^t)$ an (**Mealy-Automat**). Je nach Art der Abhängigkeit des Ausgangsvektors werden verschiedene Automatentypen unterschieden. Beim **Moore-Automaten** gilt für den Wert des Ausgangsvektors: $y^t = g(z^t)$, beim **Medvedev-Automaten** gilt $y^t = z^t$. Ein asynchroner Automat arbeitet ohne Takt.

Ein Mealy-Automat kann durch folgendes Blockschaltbild beschrieben werden:

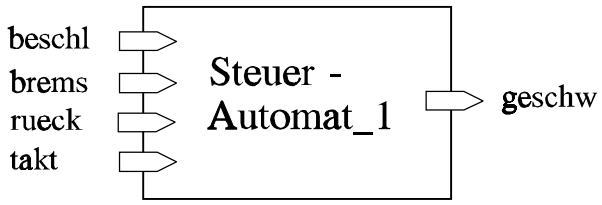


Die Überführungslogik F und die Ausgabelogik G werden als kombinatorische Schaltungen, der Zustandsspeicher unter Verwendung von Speicherelementen (Register) realisiert.

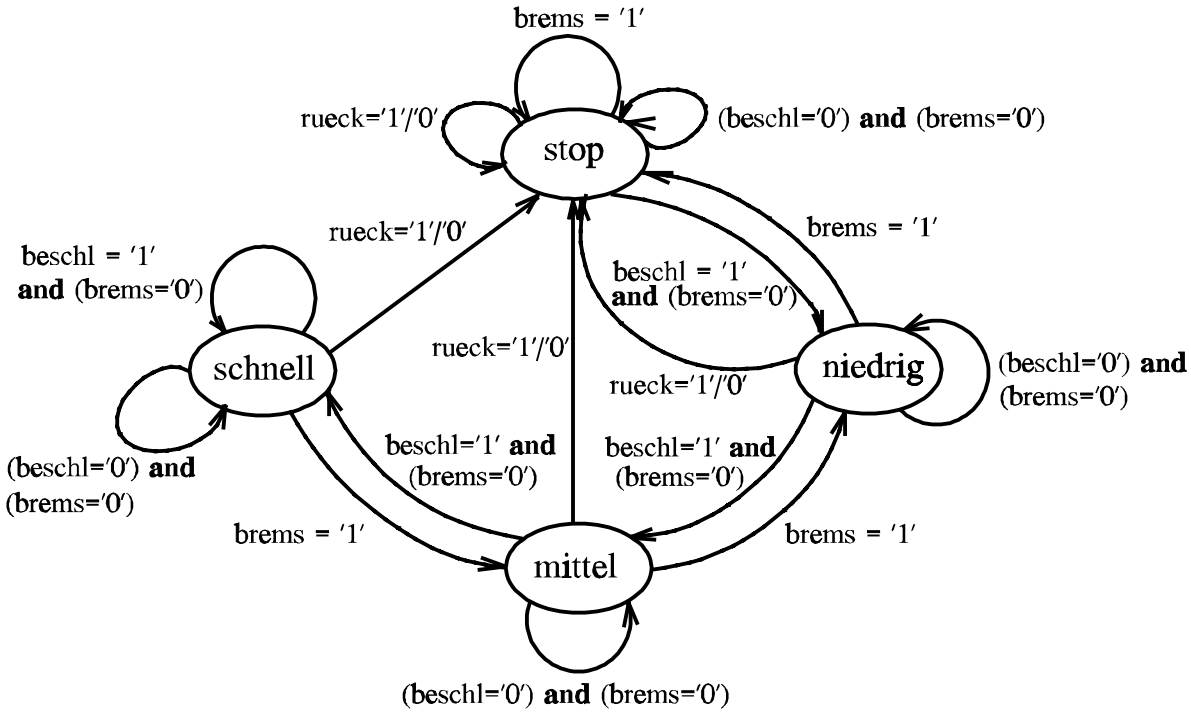
Ein erweiterter endlicher Automat enthält zusätzlich einen lokalen Datenspeicher (M charakterisiert die Menge aller im Speicher enthaltenen Werte); er führt zusätzlich die Verknüpfungsfunktion h aus; die Überführungsfunktion f und die Ausgabefunktion g sind um die Wertemenge des Speichers M zu erweitern:

Überführungsfunktion	f :	M x X x Z → Z
Ausgabefunktion	g :	M x X x Z → Y
Verknüpfungsfunktion	h :	M x X x Z → M

Beispiel: Beschreibung des Verhaltens einer Geschwindigkeitssteuerung für ein Fahrzeug. Binäre Eingangsgrößen sind "beschl" und "brems", "rueck" sowie "takt". Ausgangsgröße ist "geschw". Für beschl = ’1’ sollen der Zustand und die Geschwindigkeit synchron in der angegebenen Reihenfolge verändert werden: **stop** → **niedrig** → **mittel** → **schnell** → **schnell** → **...** . Für brems = ’1’ sollen der Zustand und die Geschwindigkeit synchron in der angegebenen Reihenfolge verändert werden: **schnell** → **mittel** → **niedrig** → **stop** → **stop** → **...** . rueck = ’1’→’0’ (fallende Flanke) soll asynchron zu "stop" führen, es besitzt eine höhere Priorität als die durch "beschl" oder "brems" ausgelösten Zustands- bzw. Geschwindigkeitsänderungen. Falls "beschl" und "brems" beide gleichzeitig mit ’1’ belegt sind, hat "brems" Priorität.



Das durch die Aufgabenstellung vorgegebene Verhalten der Geschwindigkeitssteuerung kann mit einem Automatengraphen beschrieben werden:



Da der Zustand und die Ausgangsgröße "geschw" immer den gleichen Wert haben, handelt es sich um einen Medvedev-Automat.

Vor der Beschreibung der Schnittstellen werden in einer Package typen die Werte der Ausgangsgröße "geschw" für zwei Varianten definiert: als Aufzähltyp und als Bit-Vektor (die zweite Variante ist die Basis für eine Realisierung als digitale Schaltung).

```
package typen is
  type geschw_aufz_werte is
    (stop,niedrig,mittel,schnell);
  subtype geschw_bin_werte is bit_vector (1 downto 0);
    -- Kodierung "00" - stop
    --           "01" - niedrig
    --           "10" - normal
    --           "11" - schnell
end typen;
```

```
use work.typen.all;

entity steuer_automat_1 is
  port (beschl : in bit;
        brems  : in bit;
        takt   : in bit;
        rueck  : in bit;
        geschw : buffer geschw_aufz_werte := stop);
end steuer_automat_1;

architecture verh of steuer_automat_1 is
begin
  process
  begin
    wait until (takt’event and takt = ’1’) or
                (rueck’event and rueck = ’0’);
    if rueck’event and rueck = ’0’ then
      geschw <= stop;
    elsif brems = ’1’ then
      case geschw is
        when stop    => null;
        when niedrig => geschw <= stop;
        when mittel  => geschw <= niedrig;
        when schnell => geschw <= mittel;
      end case;
    elsif beschl = ’1’ then
      case geschw is
        when stop    => geschw <= niedrig;
        when niedrig => geschw <= mittel;
        when mittel  => geschw <= schnell;
        when schnell => null;
      end case;
    else
      null;
    end if;
  end process;
end verh;
```

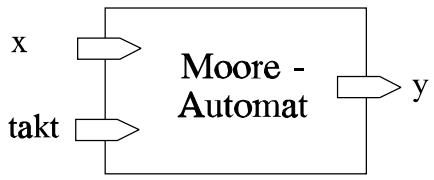
Als Zustandsspeicher wird hier das Ausgangssignal "geschw" (mit einem Aufzähltyp definiert) verwendet. Das ist möglich, weil es sich einerseits um einen Automat vom Medvedev-Typ handelt und andererseits "geschw" der Modus **buffer** zugeordnet wurde, um das Lesen dieses Objektes innerhalb der Beschreibung ("**case** geschw **is**"; siehe "Weitere VHDL-Konstrukte") zu ermöglichen.

Der Prozeß wird durch eine ’0’-’1’-Flanke des Signals "takt" oder eine ’1’-’0’-Flanke des Signals "rueck" aktiviert. Durch die **if**-Anweisung mit ihren **elsif**-Zweigen werden die in der Aufgabenstellung vorgegebenen Prioritäten realisiert: "rueck" hat Vorrang vor "brems" und "beschl", "brems" hat Vorrang vor "beschl". Die Übergänge in einen Folgezustand, die in Abhängigkeit vom aktuellen Zustand und von den Werten der Signale "brems" bzw. "beschl" erfolgen, werden mittels **case**-Anweisung realisiert.

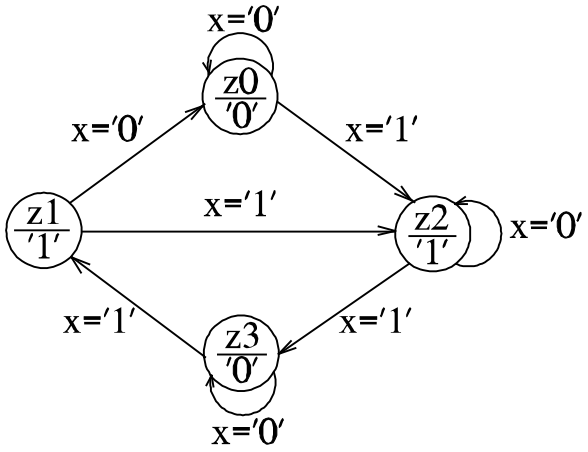
Die folgende Verhaltensbeschreibung stellt eine Modifikation der Geschwindigkeitssteuerung dar, bei der das Ausgangssignal "geschw" (und damit die internen Zustände) durch einen Bit-Vektor dargestellt werden.

```
use work.typen.all;

entity steuer_automat_2 is
  port (beschl,brems,takt,rueck : in bit;
        geschw : out geschw_bin_werte := "00");
end steuer_automat_2;

architecture verh of steuer_automat_2 is
begin
  process
    variable speed: natural := 0;
  begin
    wait until (takt'event and takt = '1') or
                (rueck'event and rueck = '0');
    if rueck'event and rueck = '0' then
      speed := 0;
    elsif brems = '1' then
      if speed > 0 then
        speed := speed - 1;
      end if;
    elsif beschl = '1' then
      if speed < 3 then
        speed := speed + 1;
      end if;
    else
      null;
    end if;
    geschw(1) <= bit'val(speed/2);
    geschw(0) <= bit'val(speed mod 2);
  end process;
end verh;
```

Beispiel: Beschreibung des Verhaltens eines Moore-Automaten entsprechend dem nachfolgend vorgegebenen Automatengraphen bzw. der Automatentabelle. Der Automat besitzt vier Zustände sowie ein Eingangs- und ein Ausgangssignal.



Automatengraph:



Der bei dem Moore-Automaten vom Zustand z^t abhängende Wert des Ausgangssignals $y^t = g(z^t)$ wurde in die die Zustände repräsentierenden Knoten aufgenommen.

Automatentabelle:

Aktueller Zustand	Folgezustand		Ausgang
	x = ’0’	x = ’1’	y
z0	z0	z2	’0’
z1	z0	z2	’1’
z2	z2	z3	’1’
z3	z3	z1	’0’

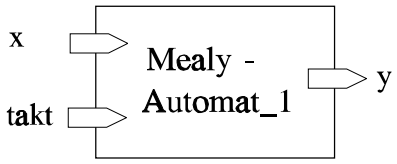
```
entity moore_automat is
  port (x,takt : in  bit;
        y      : out bit);
end moore_automat;

architecture verh of moore_automat is
  type zustand is (z0,z1,z2,z3);
  signal akt_zustand   : zustand := z0;
  signal folge_zustand : zustand;
begin
  zustandsspeicher: process
  begin
    wait until takt = ’1’;
    akt_zustand <= folge_zustand;
  end process zustandsspeicher;
```

```
    ueberfuehrungsfunktion: process (x,akt_zustand)
    begin
      case akt_zustand is
        when z0|z1 => if x = '0' then
                        folge_zustand <= z0;
                      else
                        folge_zustand <= z2;
                      end if;
        when z2    => if x = '0' then
                        folge_zustand <= z2;
                      else
                        folge_zustand <= z3;
                      end if;
        when z3    => if x = '0' then
                        folge_zustand <= z3;
                      else
                        folge_zustand <= z1;
                      end if;
      end case;
    end process ueberfuehrungsfunktion;
    ausgabefunktion: process (akt_zustand)
    begin
      case akt_zustand is
        when z0|z3 => y <= '0';
        when z1|z2 => y <= '1';
      end case;
    end process ausgabefunktion;
  end verh;
```

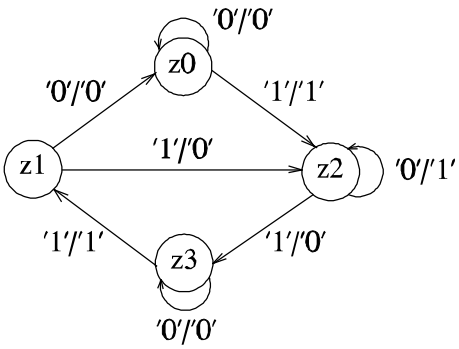
Zur Beschreibung der Zustandsübergänge werden zwei Signale eingeführt: ***akt_zustand*** für den aktuellen Zustand und ***folge_zustand*** für den aus dem aktuellen Zustand z^t und dem Eingangssignal x^t zu ermittelnden Nachfolgezustand z^{t+1}. Die Werte der Zustände werden durch den Aufzähltyp zustand definiert. Das Verhalten des Moore-Automaten wird durch drei nebenläufige Prozesse ***zustandsspeicher***, ***ueberfuehrungsfunktion*** und ***ausgabefunktion*** beschrieben. Der Prozeß ***ueberfuehrungsfunktion*** wird aktiviert durch Ereignisse des Eingangssignals ***x*** und/oder des aktuellen Zustands akt_zustand. Er berechnet für jeden, durch eine **case**-Anweisung ausgewählten aktuellen Zustand, abhängig vom Wert des Eingangssignals x den Nachfolgezustand und weist dessen Wert dem Signal ***folge_zustand*** zu. Der Prozeß ***zustandsspeicher*** steuert den Übergang zum Nachfolgezustand; er wird durch eine '0'-'1'-Taktflanke aktiviert und ersetzt den Wert des aktuellen Zustandes ***akt_zustand*** durch den für ***folge_zustand*** berechneten Wert (ein Ereignis für ***akt_zustand*** aktiviert wiederum den Prozeß ***ueberfuehrungs-funktion***). Der Prozeß ***ausgabefunktion*** wird durch Ereignisse an ***akt_zustand*** aktiviert und weist dem Signal ***y*** des Ausgabewert zu.

Beispiel: Beschreibung des Verhaltens eines Mealy-Automaten entsprechend dem nachfolgend vorgegebenen Automatengraphen bzw. der Automatentabelle. Der Automat besitzt vier Zustände sowie ein Eingangs- und ein Ausgangs-

signal.



Automatengraph:



Das Wertepaar x/y an den Knoten beschreibt das Eingangssignal x und das Ausgangssignal y.

Automatentabelle:

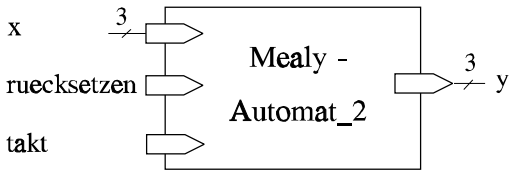
Aktueller Zustand	Folgezustand		Ausgang y	
	x = ’0’	x = ’1’	x = ’0’	x = ’1’
z0	z0	z2	’0’	’1’
z1	z0	z2	’0’	’0’
z2	z2	z3	’1’	’0’
z3	z3	z1	’0’	’1’

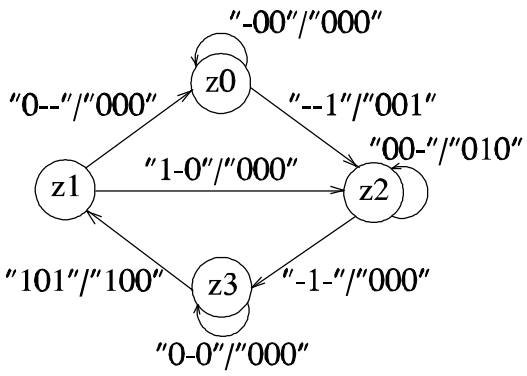
```
entity mealy_automat_1 is
  port (x,takt : in  bit;
        y      : out bit);
end mealy_automat_1;

architecture verh_1 of mealy_automat_1 is
  type zustand is (z0,z1,z2,z3);
  signal akt_zustand   : zustand := z0;
  signal folge_zustand : zustand;
```

```
begin
  synch_zustand: process
  begin
    wait until takt = '1';
    akt_zustand <= folge_zustand;
  end process synch_zustand;
  ueberf_und_ausg: process (x,akt_zustand)
  begin
    case akt_zustand is
      when z0 => if x = '0' then
                   folge_zustand <= z0;
                   y <= '0';
                 else
                   folge_zustand <= z2;
                   y <= '1';
                 end if;
      when z1 => if x = '0' then
                   folge_zustand <= z0;
                   y <= '0';
                 else
                   folge_zustand <= z2;
                   y <= '0';
                 end if;
      when z2 => if x = '0' then
                   folge_zustand <= z2;
                   y <= '1';
                 else
                   folge_zustand <= z3;
                   y <= '0';
                 end if;
      when z3 => if x = '0' then
                   folge_zustand <= z3;
                   y <= '0';
                 else
                   folge_zustand <= z1;
                   y <= '1';
                 end if;
    end case;
  end process ueberf_und_ausg;
end verh_1;
```

Die Verhaltensbeschreibung entspricht im wesentlichen der Beschreibung des oben behandelten Moore-Automaten. Folgender Unterschied besteht: Bei der innerhalb der case-Anweisung für einen aktuellen Zustand durchgeführten Berechnung werden die Werte **sowohl** für das Ausgangssignal y **als auch** für den Nachfolgezustand folge_zustand **abhängig vom Wert des Eingangssignals x** berechnet (beim Moore-Automaten wurde **nur** der Wert für folge_zustand abhängig von x berechnet).

Beispiel: Beschreibung des Verhaltens eines Mealy-Automaten entsprechend dem nachfolgend vorgegebenen Automatengraphen bzw. der Automatentabelle. Der Automat besitzt vier Zustände, drei Eingangssignale x, ein Rücksetzsignal und drei Ausgangssignale y. Das Rücksetzen soll asynchron, mit fallender 1-0-Flanke und mit höchster Priorität erfolgen.





Der Wert ’-’ bei den Signalen x bzw. y bedeutet, daß sowohl der Wert ’0’ als auch der Wert ’1’ zulässig ist (don’t care).

Automatentabelle:

Aktueller Zustand	Folgezustand				Ausgang y			
	x	z	x	z	x	y	x	y
z0	-00	z0	--1	z2	-00	000	--1	001
z1	0--	z0	1-0	z2	0--	000	1-0	000
z2	00-	z2	-1-	z3	00-	010	-1-	000
z3	0-0	z3	101	z1	0-0	000	101	100

```
entity mealy_automat_2 is
  port (x           : in  bit_vector(1 to 3);
        takt,ruecks : in  bit;
        y           : out bit_vector(1 to 3));
end mealy_automat_2;
```

```
architecture verh_1 of mealy_automat_2 is
  type zustand is (z0,z1,z2,z3);
  signal akt_zustand   : zustand := z0;
  signal folge_zustand : zustand;
begin
  synch_zustand: process (takt,ruecks)
  begin
    if ruecks’event and ruecks = ’0’ then
      akt_zustand <= z0;
    elsif takt’event and takt = ’1’ then
      akt_zustand <= folge_zustand;
    end if;
  end process synch_zustand;
  ueberf_und_ausg: process (x,akt_zustand)
  begin
    case akt_zustand is
      when z0 => if (x and "011") = "000" then
                   folge_zustand <= z0; y <= "000";
                 elsif (x and "001") = "001" then
                   folge_zustand <= z2; y <= "001";
                 else
                   folge_zustand <= akt_zustand;
                   y <= "111";
                 end if;
      when z1 => if (x and "100") = "000" then
                   folge_zustand <= z0; y <= "000";
                 elsif (x and "101") = "100" then
                   folge_zustand <= z2; y <= "000";
                 else
                   folge_zustand <= akt_zustand;
                   y <= "111";
                 end if;
      when z2 => if (x and "110") = "000" then
                   folge_zustand <= z2; y <= "010";
                 elsif (x and "010") = "010" then
                   folge_zustand <= z3; y <= "000" ;
                 else
                   folge_zustand <= akt_zustand;
                   y <= "111";
                 end if;
     when z3 => if (x and "101") = "000" then
                   folge_zustand <= z3; y <= "000";
                 elsif (x and "111") = "101" then
                   folge_zustand <= z1; y <= "100";
                 else
                   folge_zustand <= akt_zustand;
                   y <= "111";
                 end if;
    end case;
  end process ueberf_und_ausg;
end verh_1;
```

Die Beschreibung des Verhaltens ist an die Beschreibung des Mealy-Automaten im vorangegangenen Beispiel angelehnt. Der Prozeß ueberf_und_ausg berechnet für den durch eine **case**-Anweisung ausgewählten aktuellen Zustand akt_zustand den Wert des Nachfolgezustandes folge_zustand und des Ausgabevektors y, jeweils in Abhängigkeit vom Wert des Eingabevektors x. Die Bedingung in der **if**-Anweisung besteht aus einer UND-Verknüpfung des Eingangsvektors x mit einer Bitketten-Konstanten; letztere dient als Schablone, um den Teil des Eingangsvektors x auszuwählen, der laut Automatengraph bzw. Automatentabelle für die Berechnung des Nachfolgezustandes bzw. der Ausgangssignale maßgebend ist.

Der Prozeß synch_zustand wird durch Ereignisse von takt und ruecks aktiviert. Gesteuert durch eine **if**-Anweisung wird durch eine ’1’-’0’-Flanke von ruecks akt_zustand auf z0 gesetzt. Eine ’0’-’1’-Flanke des Taktes weist akt_zustand den Wert des durch den Prozeß ueberf_und_ausgabe berechneten Wertes für folge_zustand zu.

Legt man für einen Mealy-Automat die in "Sequentielle Schaltungen 47" dargestellte Struktur zugrunde, die aus den Teilen Zustandsspeicherfunktion, Überführungsfunktion und Ausgabefunktion besteht, dann kann das Verhalten eines Mealy-Automaten folgendermaßen in einer VHDL-ähnlichen Notation beschrieben werden:

```
entity mealy_automat is
  port (eing        : in  bit_vector;
        ruecks,takt : in  bit;
        ausg        : out bit_vector);
end mealy_automat;

architecture verh of mealy_automat is
  type zustands_werte is (zustand1,zustand2,...);
  signal akt_zustand,folge_zustand : zustands_werte;
begin

  zust_speicher_fkt: process (takt,ruecks)
  begin
    if ruecks = ’1’ then
      akt_zustand <= zustand1; -- Initialzustand
    elsif takt’event and takt = ’1’ and takt’last_value = ’0’ then
      akt_zustand <= folge_zustand;
    end if;
  end process zust_speicher_fkt;

  ueberf_funktion: process (akt_zustand,eing)
  begin
   case akt_zustand is
     when zustand1 => if eing = ... then
                        folge_zustand <= ...;
                      elsif eing = ... then
                        ...
                      end if;
     when zustand2 => ...
     ...
   end case;
 end process ueberf_funktion;
```

```
   ausgabe_funktion: process (akt_zustand,eing)
   begin
     case akt_zustand is
       when zustand1 => if eing = ... then
                          ausg <= ...;
                        elsif eing = ... then
                          ...
                        end if;
       when zustand2 => ...
       ...
     end case;
   end process ausgabe_funktion;
 end verh;
```

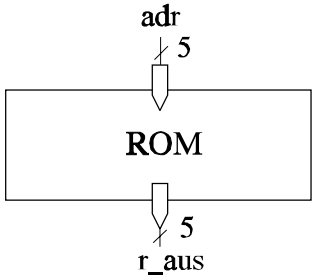
Bei der Modellierung **erweiterter endlicher Automaten** wird die Verknüpfungsfunktion in der Regel zusammen mit der Überführungsfunktion in einem einzigen Prozeß beschrieben.

Die Beschreibung der Überführungsfunktion und der Ausgabefunktion von Automaten, die in den vorangegangenen Beispielen in einer **case**-Anweisung unter Verwendung einer **if**-Anweisung und von Wertzuweisungen erfolgt, kann auch mit einem ROM realisiert werden. Der ROM muß unter seinen verschiedenen Adressen jeweils eine Bitkette mit zwei Teilen enthalten, ein Teil beschreibt einen Nachfolgezustand z^{t+1}, der andere Teil enthält den Ausgabevektor y^t. Die Adresse, die zur Auswahl einer Bitkette im ROM dient, besteht ebenfalls aus zwei Teilen: der Belegung eines Eingangsvektors x^t und dem Wert des aktuellen Zustandes z^t. Diese Vorgehensweise hat den Vorteil, daß die Veränderung des Verhaltens eines Mealy-Automaten durch Austausch einer ROM-Belegung erfolgen kann. Die Verwendung einer PLA zur Beschreibung der Überführungs- und Ausgabefunktion eines Automaten ist ggf. noch günstiger, weil nichtrelevante Eingangswerte (don't cares) für Vereinfachungen genutzt werden können.

Für das bereits behandelte Beispiel Mealy_Automat_2 (Sequentielle Schaltungen 55) ist die ROM-Belegung, die die vollständige Überführungs- und Ausgabefunktion enthält, in der nachfolgenden Tabelle dargestellt:

ROM - Adresse		ROM - Ausgang		ROM - Adresse		ROM - Ausgang	
z^t	x^t	z^{t+1}	y^{t+1}	z^t	x^t	z^{t+1}	y^{t+1}
00	000	00	000	10	000	10	010
00	100	00	000	10	001	10	010
00	001	10	001	10	010	11	000
00	011	10	001	10	011	11	000
00	101	10	001	10	110	11	000
00	111	10	001	10	111	11	000
01	000	00	000	11	000	11	000
01	001	00	000	11	010	11	000
01	010	00	000	11	101	01	100
01	011	00	000				
01	100	10	000				
01	110	10	000				

Beispiel: Beschreibung eines ROM für die Überführungs- und Ausgabefunktionen des Mealy_Automaten_2.




```
entity rom_2 is
  port (adr   : in  bit_vector (1 to 5);
        r_aus : out bit_vector (1 to 5));
end rom_2;

architecture verh of rom_2 is
begin
  process (adr)
  begin
    case adr is
      when "00000" => r_aus <= B"00_000";
      when "00001" => r_aus <= B"10_001";
      when "00011" => r_aus <= B"10_001";
      when "00100" => r_aus <= B"00_000";
      when "00101" => r_aus <= B"10_001";
      when "00111" => r_aus <= B"10_001";
      when "01000" => r_aus <= B"00_000";
```

```
        when "01001" => r_aus <= B"00_000";
        when "01010" => r_aus <= B"00_000";
        when "01011" => r_aus <= B"00_000";
        when "01100" => r_aus <= B"10_000";
        when "01110" => r_aus <= B"10_000";
        when "10000" => r_aus <= B"10_010";
        when "10001" => r_aus <= B"10_010";
        when "10010" => r_aus <= B"11_000";
        when "10011" => r_aus <= B"11_000";
        when "10110" => r_aus <= B"11_000";
        when "10111" => r_aus <= B"11_000";
        when "11000" => r_aus <= B"11_000";
        when "11010" => r_aus <= B"11_000";
        when "11101" => r_aus <= B"01_100";
        when others  => r_aus <= B"11_111";
      end case;
    end process;
  end verh;
```

Die Beschreibung des Mealy-Automaten mit den im rom_2 definierten Überführungs- und Ausgabefunktionen erfordert das Hinzufügen eines Registers als Zustandsspeicher; das Register dient gleichzeitig zur Zwischenspeicherung der Ausgangswerte. Der entstehende ROM-Controller hat folgende Struktur: