

HARDWARE- BESCHREIBUNGSSPRACHEN

Hardwareentwurf mit VHDL

21. Oktober 2021
Revision: b941727 (2021-01-16 01:57:51 +0100)

Steffen Reith

Theoretische Informatik
Studienbereich Angewandte Informatik
Hochschule **RheinMain**



Notizen

Notizen

ENDLICHE AUTOMATEN MIT VHDL

ENDLICHE AUTOMATEN MIT VHDL

Ein **endlicher Automat** (engl. >finite state machine<, kurz: **FSM**) modelliert ein bestimmtes Verhalten mit Hilfe einer (endlichen) Menge von **Zuständen** und mit **Übergängen** zwischen diesen.

Im Gegensatz zu den einfachen endlichen Automaten werden im Hardwareentwurf so genannte **Transducer**, verwendet die mit Hilfe der **Eingabe** und des **aktuellen Zustands** die jeweiligen **Ausgaben generieren**.

Man unterscheidet **Moore-** und **Mealy-Automaten**, wobei man die Mealy-Automaten als Verallgemeinerung der Moore-Automaten auffassen kann.

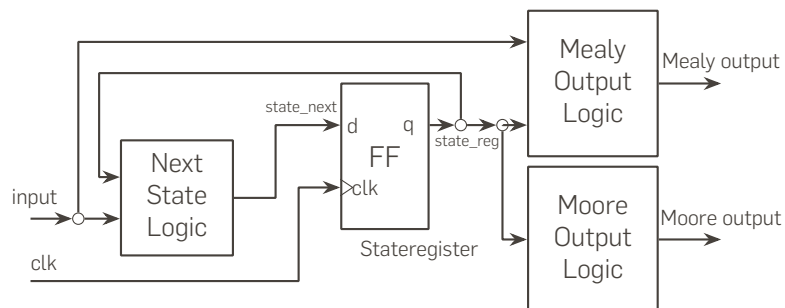
- **Moore**-Automat: Die Ausgabe wird **nur** in Abhängigkeit vom aktuellen Zustand erzeugt.
- **Mealy**-Automat: Die Ausgabe wird in Abhängigkeit von Zustand **und** aktueller Eingabe erzeugt.

103

Notizen

DIE STRUKTUR EINES SYNCHRONEN FSM

Synchrone FSMs sind idealisiert wie folgt aufgebaut:



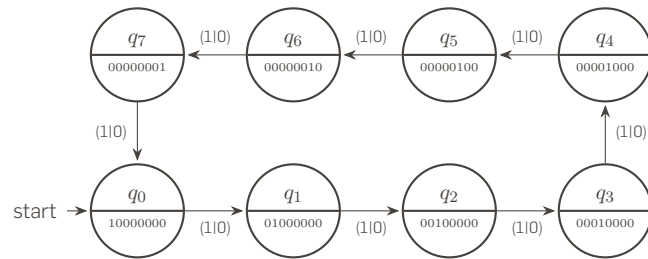
Die obige Struktur zeigt, dass Moore-Automaten eine zum **Takt synchrone Ausgabe** erzeugen.

In der Praxis werden allerdings auch Mischformen von Moore- und Mealy-Automaten verwendet.

104

Notizen

BEISPIEL: MOORE-AUTOMAT / LAUFLICHT



Dieser Automat wechselt bei jeder steigenden Flanke in den nächsten Zustand. Aus diesem Grund sind die Übergänge mit **jeder möglichen** Eingabe markiert.

Der in der **unteren** Hälfte der Zustände enthaltene Bitvektor entspricht der **Ausgabe** des Automaten ($\hat{=}$ Zustand der acht LEDs).

Notizen

105

DAS LAUFLICHT IN VHDL

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity lights is
5      port (clk, reset : in std_logic;
6            leds       : out std_logic_vector(7 downto 0));
7  end lights;
8
9  architecture Behavioral of lights is
10
11     type state_t is (q0, q1, q2, q3, q4, q5, q6, q7);
12     signal state_reg, state_next : state_t;
13
14     begin
15         transition : process (clk, reset)
16         begin
17             if (reset = '1') then
18                 state_reg <= q0; -- set initial state
19             elsif (rising_edge(clk)) then -- changes on rising edge
20                 state_reg <= state_next;
21             end if;
22         end process;

```

Notizen

106

DAS LAUFLICHT IN VHDL (II)

```

1  next_state_proc : process (state_reg)
2  begin
3      case state_reg is
4
5          when q0 =>
6              leds <= "10000000"; -- Moore Ausgabe
7              state_next <= q1;
8
9          when q1 =>
10             leds <= "01000000"; -- Moore Ausgabe
11             state_next <= q2;
12
13             ....
14
15         end case;
16     end process;

```

Im obigen Beispiel muss **nicht** mit `if-elsif-else-end if` gearbeitet werden, um den **Nachfolgezustand zu ermitteln**, da der Automat an jeder steigenden Taktflanke einfach zum nächsten Zustand übergeht.

107

Notizen

NEXT STATE LOGIK FÜR MOORE-AUTOMATEN

Im **allgemeinen Fall** wird ein Moore-Automat für die Eingaben noch einen **input-Port** haben. Dann ist `input` in der Sensitivitätsliste von `next_state_proc` enthalten.

Damit ergibt sich folgendes Template:

```

1  next_state_proc : process (state_reg, input)
2  begin
3      case state_reg is
4          when state_a =>
5              output <= <value>; -- Moore Ausgabe
6              if (input = <value>) then -- Alle Möglichkeiten abfragen!
7                  state_next <= state_<num>;
8                  ...
9              else
10                 state_next <= state_<num>;
11             end if;
12         when state_b =>
13             ....

```

108

Notizen

NEXT STATE LOGIK FÜR MOORE-AUTOMATEN (II)

Hinweis: `next_state_proc` ist ein **kombinatorischer Schaltkreis**. Also wird `output` in jedem Zweig belegt und es werden alle Möglichkeiten zur Bestimmung von `next_state` berücksichtigt. Damit ergeben sich drei Designregeln:

- Spezifiziere alle Möglichkeiten im `next_state_proc`, d.h. im `case`-Statement werden **alle Bedingungen** der jeweiligen `if`-Anweisung aufgeführt. Die `case`-Anweisung berücksichtigt **alle möglichen Zustände** des Automaten. Evtl. kann man mit „don't cares“ arbeiten. Wird dies nicht gemacht, so kann der Synthesizer ungewollt Latches einführen.
- Signalzuweisungen sind **nur im jeweiligen Zustand gültig**. Zuweisungen werden nicht gespeichert (kombinatorischer Prozess)!

109

Notizen

NEXT STATE LOGIK FÜR MOORE-AUTOMATEN (III)

- Müssen Werte **gespeichert** (\triangleq registered) werden, so wird dies mit Signalen der Form `XXX_reg` bzw. `XXX_next` **im Prozess transition** realisiert. Grund: Speicher / FlipFlops werden nur im sequentiellen Prozess `transition` verwendet.

Notizen

110

ALTERNATIVE (LAUFLICHT): SEPARATE PROZESSE

Beobachtung: Ist das State-Diagramm erstellt, so ist die Umsetzung in eine Implementierung nahezu mechanisch.

Ist der Automat verhältnismäßig groß oder unübersichtlich, so kann man **Ausgabe** und **next-state Logik** auch trennen.

Next-state Logik:

```

1  next_state_proc : process (state_reg)
2  begin
3      case state_reg is
4          when q0 =>
5              state_next <= q1;
6          when q1 =>
7              state_next <= q2;
8
9              ....
10
11         end case;
12     end process;

```

111

Notizen

ALTERNATIVE: SEPARATE PROZESSE (II)

Output Logik:

```

1  output_proc : process (state_reg)
2  begin
3      case state_reg is
4          when q0 =>
5              leds <= "10000000"; -- Moore Ausgabe
6          when q1 =>
7              leds <= "01000000"; -- Moore Ausgabe
8
9              ....
10
11         end case;
12     end process;

```

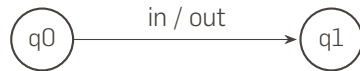
Größere Automaten kann man auch mit speziellen graphischen Tools beschreiben, die dann den benötigten VHDL-Code erzeugen.

112

Notizen

MEALY - AUTOMATEN

Ein Mealy-Automat ermittelt den Output mit Hilfe des Zustandes **und** der Eingabe. Aus diesem Grund werden die **Übergänge** eines Mealy-Automaten zusätzlich mit der Ausgabe beschriftet.



Dabei bedeutet die **Kantenbeschriftung in / out**: Zustandswechsel von q0 nach q1, wenn die Eingabe in vorliegt. Gebe bei dem Zustandswechsel out aus.

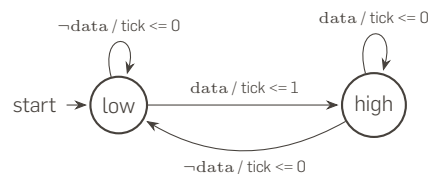
Aufgrund der Struktur eines Mealy-Automaten muss die Änderung der Ausgabe **nicht synchron** zum Takt sein (z.B. wenn sich die Eingabesignale ändern, so dass die Ausgabe den Pegel wechselt).

113

Notizen

BEISPIEL: MEALY - AUTOMATEN

Der folgende Automat erkennt **steigenden Flanken** in dem Signal **data**:



```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity edgeDetect is
5   port (clk   : in  std_logic;
6         reset : in  std_logic;
7         data  : in  std_logic;
8         tick  : out std_logic);
9 end edgeDetect;
  
```

114

Notizen

BEISPIEL: MEALY - AUTOMATEN (II)

```

1  architecture mealy of edgeDetect is
2      type state_t is (low, high);
3      signal state_reg : state_t;
4      signal state_next : state_t;
5  begin
6
7      transition : process (clk, reset)
8      begin
9          if (reset = '1') then
10             state_reg <= low;
11         elsif (rising_edge(clk)) then
12             state_reg <= state_next;
13         end if;
14     end process;
15
16     -- next-state / output logic
17     next_state_proc : process(state_reg, data)
18     begin
19         -- Set default values
20         state_next <= state_reg;
21         tick <= '0';

```

115

Notizen

BEISPIEL: MEALY - AUTOMATEN (II)

```

1  case state_reg is
2
3      -- Handle state 'low'
4      when low =>
5          if (data = '1') then
6              state_next <= high;
7              tick <= '1';
8          end if;
9
10     -- Handle state 'high'
11     when high =>
12         if (data = '0') then
13             state_next <= low;
14         end if;
15
16     end case;
17
18 end process;
19 end architecture;

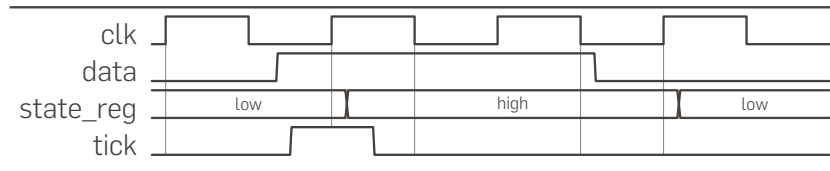
```

116

Notizen

TIMING DIAGRAMM DES MEALY-AUTOMATEN

Es wurde schon erwähnt, dass die Ausgabe eines Mealy-Automaten evtl. **nicht synchron** zum Takt sein kann:



Durch dieses Verhalten werden auch kurzzeitige Signalschwankungen (engl. **glitch(es)**) der Eingabe evtl. als Ausgabe weitergeleitet. Dies kann zu Problemen führen.

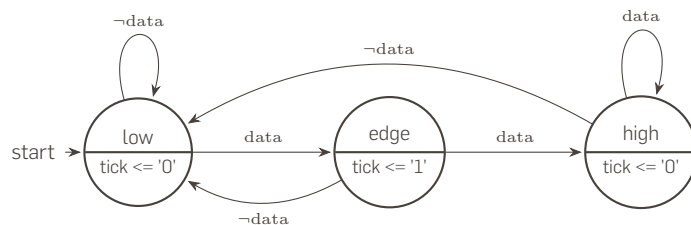
Auch die **Breite** des Signals **tick** kann **variieren**, je nachdem wie die Lage der steigenden Flanke von **data** zu **clk** ist.

117

Notizen

DIE FLANKENERKENNUNG MIT EINEM MOORE-AUTOMAT

Eine ähnliche Flankenerkennung kann man mit Hilfe eines Moore-Automaten entwerfen:



Die übliche Codierung eines Moore-Automaten ergibt:

```

1 architecture moore of edgeDetect is
2   type state_t is (low, edge, high);
3   signal state_reg : state_t;
4   signal state_next : state_t;
5 begin
6   -- Prozess stateHandler wie im Mealy-Fall

```

118

Notizen

DIE FLANKENERKENNUNG MIT EINEM MOORE-AUTOMAT (II)

```

1  next_state_proc : process(state_reg, data)
2  begin
3
4      -- Set default values
5      state_next <= state_reg;
6      tick <= '0';
7
8      case state_reg is
9
10         when low =>
11             if (data = '1') then
12                 state_next <= edge;
13             end if;
14
15         when edge =>
16             tick <= '1';
17             if (data = '1') then
18                 state_next <= high;
19             else
20                 state_next <= low;
21             end if;

```

119

Notizen

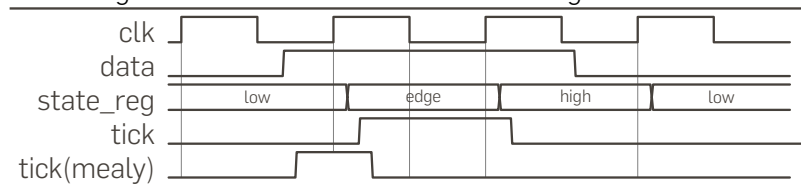
DIE FLANKENERKENNUNG MIT EINEM MOORE-AUTOMAT (III)

```

1      when high =>
2          if (data = '0') then
3              state_next <= low;
4          end if;
5      end case;
6  end process;
7  end architecture;

```

Damit ergibt sich ein leicht verändertes Timing:



Es zeigt sich, dass der von einem Moore-Automaten generierte Tick **immer einen clk-Zyklus** lang ist.

120

Notizen

MOORE VS. MEALY

Die Ausgabe von Moore-Automaten ist **synchron**, wogegen die von Mealy-Automaten auch **nicht synchron** sein kann.

Die Ausgabe von Moore-Automaten **ändert sich nur an Taktflanken** und ist so **robuster** gegen Glitches. Die Reaktion von Mealy-Automaten ist aber evtl. schneller.

Nachteil: Moore-Automaten haben normalerweise **mehr Zustände** als gleichwertige Mealy-Automaten und brauchen somit **mehr Fläche** für die Logik.

Am Timing-Diagramm sieht man, dass das Signal dieses **Mealy**-Automaten **einen Takt früher** verfügbar ist. Wird tick in einem synchronen Subsystem weiter verwendet, so spielen Glitches und die Asynchronität eine untergeordnete Rolle, da das Signal nur an der **steigenden Flanke stabil** sein muss.

121

Notizen

Notizen
