5. Zustandsautomaten

5.1 Der abstrakte Automat der Automatentheorie

Der abstrakte Automat[1)] ist eine ganz allgemeine, naheliegende Modellvorstellung, die alles umfasst, was man irgendwie mit dem Begriff des automatischen (selbsttätigen) Arbeitens in Verbindung bringen kann. Er wird durch drei Mengen und zwei Funktionen gekennzeichnet. Abstrakte Automaten arbeiten zu diskreten Zeitpunkten (Taktzeitpunkten). Es gibt also einen Zeitpunkt t (den aktuelle Zeitpunkt), t + 1 (den darauf folgende Zeitpunkt) usw. (t – 1, t – 2 usw. bezeichnen vorhergegangene Zeitpunkte, also die Vergangenheit oder Vorgeschichte). Wie diese Zeitpunkte definiert werden (durch Taktflanken, durch das Erkennen von Änderungen usw.), ist der Theorie gleichgültig.

Die Mengen:

- **E** = {E_1, E_2,...} Menge der Eingangssignale (Eingangsalphabet),
- **A** = {A_1, A_2,...} Menge der Ausgangssignale (Ausgangsalphabet),
- **Z** = {Z_1, Z_2,...} Menge der Zustände.

Die Funktionen:

- Überführungsfunktion f. Sie beschreibt, welchen Zustand der Automat im nächsten diskreten Zeitpunkt (t+1) einnehmen wird (Folgezustand):

$$Z\,(t + 1) = f\,(\text{irgendwas})$$

- Ausgabefunktion g. Sie beschreibt, welche Ausgangsbelegung der Automat abgibt:

$$A\,(t) = g\,(\text{irgendwas})$$

Das “irgendwas” im Funktionsausdruck bestimmt den Unterschied. Demgemäß gibt es zwei Grundtypen von Automaten, den Mealy-Automaten und den Moore-Automaten. Hinzu kommt ein Automatentyp ohne Ausgabefunktion, der Semiautomat oder Medwedjew-Automat. Er wird dann verwendet, wenn nur die Zustände von Bedeutung sind.

1: Genauer: der initiale deterministische endliche Automat. Seine Merkmale: (1) der Automat beginnt seine Arbeit immer mit einem einzigen bestimmten Anfangszustand, (2) gleiches Eingangssignal und gleicher Zustand führen stets zum gleichen Ausgangssignal und zum gleichen Folgezustand, (3) die Anzahl aller Eingangssignale, Ausgangssignale und Zustände ist endlich.

Zu jedem beliebigen Taktzeitpunkt gilt:

- Der Automat hat *einen* Zustand.
- Es liegt *ein* Eingangssignal an.
- Es wird *ein* Ausgangssignal abgegeben.

Die Signale und Zustände zu einem beliebigen Taktzeitpunkt t sind Elemente der jeweiligen Menge:

- Eingangssignal: $E(t) = \in \mathbf{E}$.
- Ausgangssignal: $A(t) = \in \mathbf{A}$.
- Zustand: $Z(t) = \in \mathbf{Z}$.

Das Wort
Ein "Wort" im Sinne der Automatentheorie ist eine Folge von Signalen oder Zuständen in aufeinanderfolgenden Taktzeitpunkten:

- Eingangswort ($E(t_1)$, $E(t_2)$, $E(t_3)$ usw.).
- Ausgangswort ($A(t_1)$, $A(t_2)$, $A(t_3)$ usw.).
- Zustandswort ($Z(t_1)$, $Z(t_2)$, $Z(t_3)$ usw.).

Der Anfangszustand
Damit die Betrachtungen nicht ins Uferlose führen, wird im Folgenden stets vorausgesetzt, dass sich der Automat zu Beginn der Arbeit (= zum Taktzeitpunkt t_0) in einem bestimmten Anfangszustand $Z(t_0) \in \mathbf{Z}$ befindet (initialer Automat).

Der abstrakte Automat als Black Box
Nur der Eingang kann von außen mit Signalen belegt, nur der Ausgang beobachtet werden. Der Zustand ist nicht sichtbar.

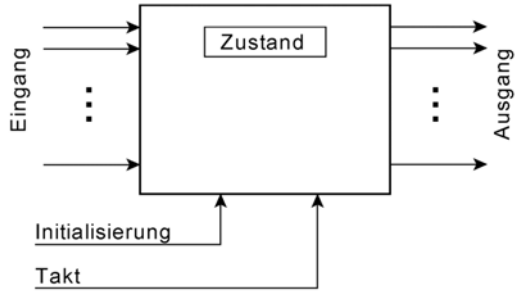

Abb. 5.1 Der abstrakte Automat als Black Box. Das Initialisierungssignal versetzt den Automaten in seinen Anfangszustand, das Taktsignal gibt die diskreten Zeitpunkte vor.

Gleichartigkeit (Äquivalenz) von Automaten
Zwei Automaten gelten als gleichartig, wenn sie das Gleiche leisten, also – vom jeweiligen Anfangszustand ausgehend – auf gleiche Wortfolgen am Eingang gleiche Wortfolgen am Ausgang abgeben. Wie viele Zustände und Zustandsübergänge sie hierzu benötigen, ist gleichgültig. Nutzanwendung: Man kann Automatenfunktionen auf verschiedene Weise implementieren, um Vorgaben in Hinsicht auf Geschwindigkeit und Kosten zu erfüllen. Typische Auslegungen:

a) So wenige Zustände wie möglich, wobei eine komplexe Überführungsfunktion in Kauf genommen wird.
b) Mehr Zustände, aber eine einfachere Überführungsfunktion.

Geht es um geringste Kosten, ist offensichtlich die Auslegung b) zu bevorzugen, vor allem in Form speicherbasierter oder programmseitige Implementierungen (z. B. mittels Mikrocontroller). Geht es um höchste Geschwindigkeit, so hängt die Entscheidung zwischen a) und b) von der jeweiligen Entwurfsaufgabe und der einzusetzenden Schaltkreistechnologie ab. Die Auslegung b) ist dann überlegen, wenn sich ein Zustandsübergang a) mit einer Taktzykluszeit t_a durch n Zustandsübergänge b) ersetzen lässt, von denen jeder in einer Taktzykluszeit $t_b < t_a : n$ ausgeführt werden kann. Ggf. muss die Entscheidung anhand von Probeentwürfen getroffen werden. Praxistipp: Oftmals ist der “langsamere” Takt besser (Stromaufnahme, Störstrahlung). Auch ist gemäß a) die Vorhaltezeit nur einmal einzurechnen, gemäß b) hingegen n mal. Komplizierte Überführungsfunktionen kann man oftmals mit speicherbasierten Zuordnern erledigen.

Der Mealy-Automat

Überführungsfunktion
Der Folgezustand ergibt sich aus dem aktuellen Zustand und dem aktuellen Eingangssignal.

$$Z\,(t+1) = f\,(Z\,(t), E\,(t))$$

Ausgabefunktion
Das Ausgangssignal ergibt sich aus dem aktuellen Zustand und dem aktuellen Eingangssignal.

$$A\,(t) = g\,(Z\,(t), E\,(t))$$

Zu jedem Taktzeitpunkt geschieht Folgendes:

- Der zum vorherigen Taktzeitpunkt berechnete Folgezustand wird zum aktuellen Zustand.
- Die Eingangsbelegung wird abgetastet.
- Aus der Eingangsbelegung und dem aktuellen Zustand wird der Folgezustand berechnet. Er kommt aber nicht sofort zur Wirkung, sondern wird bis zum nächsten Taktzeitpunkt vorgemerkt.

- Aus der Eingangsbelegung und dem aktuellen Zustand wird die Ausgangsbelegung berechnet. Sie kommt sofort zur Wirkung.

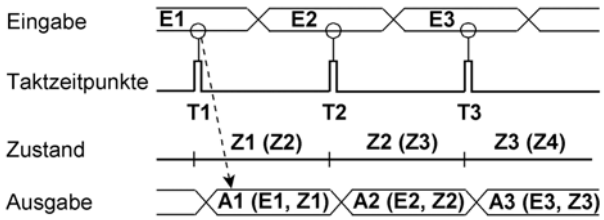

Abb. 5.2 Zur Arbeitsweise des Mealy-Automaten. Zum Taktzeitpunkt T1 wird der Zustand Z1 wirksam, und die Eingangsbelegung E1 wird abgetastet. Daraus werden der Folgezustand Z2 sowie die aktuelle Ausgangsbelegung A1 ermittelt. Die Ausgangsbelegung wird sofort wirksam, der Folgezustand hingegen erst zum nächsten Taktzeitpunkt (T2). Der Pfeil deutet an, dass sich die Eingangsbelegung E1 bereits zum Taktzeitpunkt T1 auf die Ausgangsbelegung auswirken kann.

Der Moore-Automat

Überführungsfunktion
Der Folgezustand ergibt sich aus dem aktuellen Zustand und dem aktuellen Eingangssignal.

$$Z\ (t+1) = f\ (Z\ (t), E\ (t))$$

Ausgabefunktion
Das Ausgangssignal ergibt sich aus dem aktuellen Zustand.

$$A\ (t) = g\ (Z\ (t))$$

Zu jedem Taktzeitpunkt geschieht Folgendes:

- Der zum vorherigen Taktzeitpunkt berechnete Folgezustand wird zum aktuellen Zustand.
- Die Eingangsbelegung wird abgetastet.
- Aus der Eingangsbelegung und dem aktuellen Zustand wird der Folgezustand berechnet. Er kommt aber nicht sofort zur Wirkung, sondern wird bis zum nächsten Taktzeitpunkt vorgemerkt.
- Aus dem aktuellen Zustand wird die Ausgangsbelegung berechnet. Sie kommt sofort zur Wirkung.

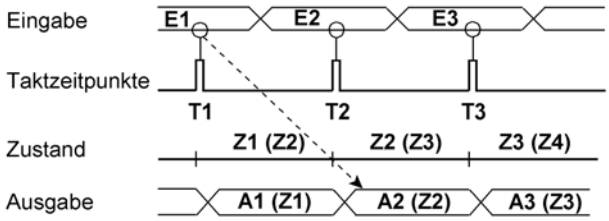

Abb. 5.3 Zur Arbeitsweise des Moore-Automaten. Zum Taktzeitpunkt T1 werden der Zustand Z1 und die zugehörige Ausgangsbelegung A1 wirksam. Die Eingangsbelegung E1 wird abgetastet. Daraus wird der Folgezustand Z2 ermittelt. Er wird zum nächsten Taktzeitpunkt T2 wirksam. Damit ergibt sich auch die Ausgangsbelegung A2. Der Pfeil deutet an, dass sich die Eingangsbelegung E1 erst zum folgenden Taktzeitpunkt T2 über den Zustand Z2 auf die Ausgangsbelegung A2 auswirken kann.

Eingänge und Ausgänge

Der Mealy-Automat kann auf die jeweilige Eingangsbelegung sofort (zum gleichen Taktzeitpunkt) reagieren, also mit einer Ausgangsbelegung antworten. Der Moore-Automat hingegen muss zunächst den Folgezustand einnehmen (die Eingangsbelegung beeinflusst den Folgezustand, dieser wiederum bestimmt die Ausgangsbelegung). Das Ausgangssignal wird also stets erst zum folgenden Taktzeitpunkt wirksam, das heißt einen Takt später als beim Mealy-Automaten.

Wechselseitige Wandlungen[2)]:

- Jeder Moore-Automat kann durch einen Mealy-Automaten interpretiert werden, der die gleiche Zustandsmenge und die gleiche Überführungsfunktion hat. Die Ausgangssignale treten dabei einen Taktzeitpunkt eher auf als beim Moore-Automaten (zeitverschobene Ausgabefunktion).
- Zu jedem Mealy-Automat läßt sich ein Moore-Automat mit gleichem Ausgangsverhalten angeben (allerdings nicht notwendigerweise mit der gleichen Zustandsmenge). Die Ausgangssignale treten dabei einen Taktzeitpunkt später auf als beim Mealy-Automaten (zeitverschobene Ausgabefunktion).

Faustregel:

Der Mealy-Automat hat weniger Zustände, aber eine komplexere Ausgabefunktion. Die Ausgangsbelegung ist von der Eingangsbelegung und vom Zustand abhängig). Der Moore-Automat hat mehr Zustände, aber eine einfachere Ausgabefunktion. Die Ausgangsbelegung ergibt sich nur aus dem Zustand – es ist im Grunde eine einfache Umcodierung.

2: Zu Beweisen und Umwandlungsverfahren sei auf die Standardwerke der Automatentheorie verwiesen.

Der Semiautomat (Medwedjew-Automat)

Überführungsfunktion
Der Folgezustand ergibt sich aus dem aktuellen Zustand und dem aktuellen Eingangssignal.

$$Z\,(t+1) = f\,(Z\,(t), E\,(t))$$

Ausgabefunktion
Gibt es nicht. Es interessieren nur die Zustände.

Autonome Automaten
Ein autonomer Automat hat keine Eingänge. Er durchläuft zyklisch eine Folge von Zuständen. Typische Beispiele sind Uhren und Taktgeber. Überführungs- und Ausgabefunktion hängen nur vom aktuellen Zustand ab:

$$Z\,(t+1) = f\,(Z\,(t)); A\,(t) = g\,(Z\,(t))$$

5.2 Reale Zustandsautomaten (State Machines)

Der Zustandsautomat (Finite State Machine FSM oder kurz State Machine) ist eine näherungsweise technische Verwirklichung des Konzepts vom abstrakten Automaten. Überführungs- und Ausgabefunktionen werden mit Schaltnetzen dargestellt (Funktionszuordner), die Zustandsspeicherung ergibt sich durch Einfügen von Flipflops in den Rückführungsweg (Zustandsregister). Die wesentlichen Unterschiede zum abstrakten Automaten:

- Man arbeitet typischerweise nicht mit Alphabeten und Worten, sondern mit binär belegten Signalleitungen und binär codierten Signaldarstellungen.
- Die tatsächlich ablaufende Zeit kann nicht ohne Weiteres vernachlässigt werden.

5.2.1 Zustandsautomaten in der Entwurfspraxis

Die Grundlagen der Automatentheorie kann man auf beliebige sequentielle Schaltungen anwenden, beginnend mit den einfachsten – den Flipflops. Hierbei ergeben sich mehrere grundsätzliche Anwendungsgebiete:

Verhaltensmodellierung und Verhaltensbeschreibung
Die Begriffe und Zusammenhänge der Automatentheorie dienen als Modellvorstellung zum Veranschaulichen, Klarmachen und Beschreiben von Entwurfsaufgaben und Entwurfsabsichten.

Entwerfen von sequentiellen Schaltungen
Das gewünschte Verhalten wird mit Zuständen, Zustandsübergängen usw. beschrieben. Diese Beschreibung wird dann auf systematische Weise in ein Schaltwerk umgesetzt. Beim heutigen

Stand der Technik ist es möglich, nur das Verhalten – als Entwurfsabsicht – formal zu beschreiben (Programmiersprache, Hardwarebeschreibungssprache) und es dem Entwicklungssystem zu überlassen, eine entsprechende Schaltung auszuarbeiten. Manche Systeme unterstützen auch die graphische Entwurfserfassung über Zustandsdiagramme.

Programmierung und Programmlogik

Der Zustandsautomat ist ein nützliches und effektives Programmiermodell vor allem für solche Aufgaben, die mit dem Reagieren auf Signalbelegungen zu tun haben.

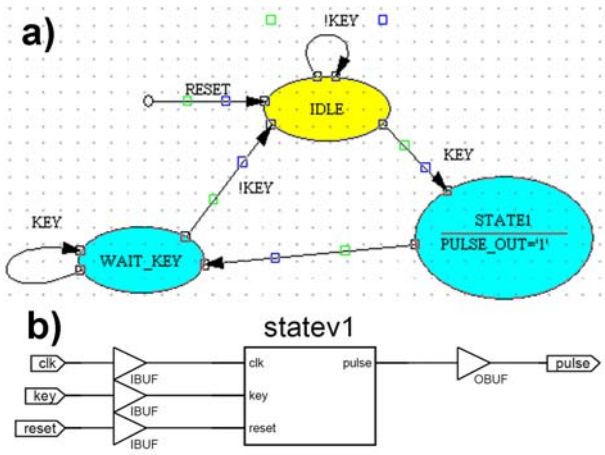

Abb. 5.4 Entwurfserfassung über Zustandsdiagramm ([S04]). a) Entwurfseingabe, b) der Zustandsautomat als Funktionselement.

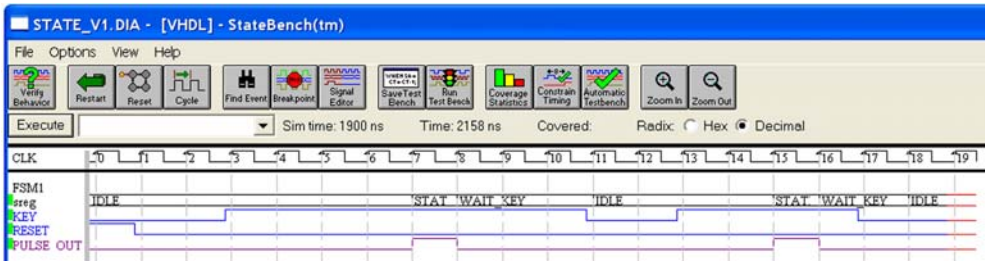

Abb. 5.5 Entwurfserfassung über Zustandsdiagramm ([S04]). Ein Simulationsergebnis.

Die Verfahren des Zustandsautomatenentwurfs beruhen auf exakten Grundlagen und einleuchtenden Begriffsbildungen. Komplizierte Entwurfsaufgaben können so weitgehend routinemäßig gelöst werden. Hierdurch werden – im Gegensatz zum rein intuitiven Vorgehen – viele Entwurfsfehler von vornherein vermieden. Andere lassen sich auf formale Weise (also rechnergestützt) erkennen. So kann das hier vorgestellte Entwicklungssystem u. a. folgende Fehler auffinden:

- Hängenbleiben in einem Zustand,
- der Folgezustand ist unbestimmt,
- Zustände, die nie erreicht werden.

Was ist eigentlich ein Zustand? – Die kombinatorische Explosion

Die Frage weist auf eine Grenze hin, die die Anwendbarkeit des an sich naheliegenden Modells des Zustandsautomaten einschränkt. Jedes Flipflop hat zwei Zustände. Eine sequentielle Schaltung, die n Flipflops enthält, kann in jedem Taktzyklus einen von insgesamt 2^n möglichen Zuständen einnehmen. Hält man sich konsequent an die Definitionen der Automatentheorie, so trägt jedes gespeicherte Bit (z. B. im RAM eines Mikrocontrollers) zur Zustandsmenge bei, und jeder einzelne Maschinentakt bewirkt einen Zustandsübergang. Dieses Anwachsen der Zustandsmenge (gemäß $O(2^n)$) und damit der Menge der möglichen Zustandsübergänge bezeichnet man auch als kombinatorische Explosion.

Das Modell des Zustandsautomaten kann deshalb nicht schematisch auf alles angewendet werden, was gespeicherte Bits aufweist[3)], sondern (1) nur mit Bedacht und (2) nur auf Probleme, die durch überschaubare Zustandsmengen oder vergleichsweise einfache Zustandsübergänge gekennzeichnet sind. Als Beispiele seien genannt:

- Eine komplexe Steuerschaltung mit 20 Zuständen.
- Eine Zählschaltung mit einer Million Zuständen (das erscheint viel, aber dafür sind die Zustandsübergänge einfach: Zustand 0 – Zustand 1 – Zustand 2 usw.).

Wie sich der Praktiker behilft

Das Modell des Zustandsautomaten ist zum allgemeinen Standard geworden. In Hinsicht auf die praktische Anwendung dieses Modells gibt es mehrere Ansätze, um das Problem der kombinatorischen Explosion zu beherrschen oder zu umgehen:

Alles nicht so verbissen sehen

Das Prinzip des Zustandsautomaten wird praktisch nutzbar gemacht, indem man es nicht so genau nimmt, also nicht jedes gespeicherte Bitmuster als Zustand und jeden Taktzyklus als Zustandsübergang auffasst. Der Zustandsautomat ist vielmehr ein eher globales Modell, das die grundsätzliche Funktionsweise beschreibt. Die Einzelheiten werden auf andere Weise erledigt. Dieses Herangehen ist vor allem in der Programmentwicklung üblich. Der Zustandsautomat beschreibt die grundsätzliche Programmlogik; die Einzelheiten (Erfassung der Eingangssignale, das Umschalten von einem Zustand zum nächsten sowie die Ausgabefunktionen) werden hingegen auf herkömmliche Weise ausprogrammiert.

Teile und herrsche

Es werden nur jene Schaltungsteile als Zustandsautomat ausgeführt, die sich dafür eignen. Das typische Beispiel ist die Aufteilung in Steuer- und Operationswerke, wobei die Steuerwerke als Zustandsautomaten entworfen werden.

3: Obwohl die reine Theorie dies zulässt – nichts spricht grundsätzlich dagegen, einen Personalcomputer mit vier Prozessorkernen, acht GBytes Arbeitsspeicher und mehreren Graphikpipelines als Zustandsautomaten anzusehen ...

Mehrere Zustandsautomaten im Verbund
Viele Entwurfsaufgaben führen von Anfang an darauf, mehrere Zustandsautomaten vorzusehen. Deren Zusammenwirken kann auf intuitive Weise, durch Nutzung formalisierter Modelle oder auf Grundlage theoretischer Ansätze organisiert werden.

Funktionelle Anreicherung
Der einfache Zustand, der lediglich erreicht, gehalten und verlassen werden kann, wird um Zusatzfunktionen erweitert, die beim Erreichen, während des Haltens und beim Verlassen ausgeführt werden. Ein solcher "dicker" Zustand ersetzt Folgen mehrerer "gewöhnlicher" Zustände. Damit wird das Zustandsmodell der jeweiligen Entwurfsaufgabe besser überschaubar.

Die Übergänge zwischen diesen Ansätzen sind fließend. Viele Teillösungen werden im Lauf der Problembearbeitung intuitiv gefunden. Manche Prinzipien werden von den Entwicklungsumgebungen unterstützt (z. B. mehrere Zustandsautomaten im Verbund). Mit Entwurfsbeschreibungssprachen können im Grunde beliebige Kombinationslösungen dargestellt werden (mehrere Zustandsautomaten, Zustandsautomaten mit ergänzenden kombinatorischen und sequentiellen Schaltungen usw.). Zudem gibt es Beschreibungsmittel, die durchgehend formalisiert sind. Beispiel: die Modellierungssprache UML (Universal Modeling Langage). Die UML-Zustandsdiagramme (State Charts) unterstützen alle vorstehend erläuterten Entwurfsansätze.

5.2.2 Grundschaltungen

Zu jedem abstrakten Automaten lassen sich äquivalente Zustandsautomaten angeben. Die abstrakten Signale und Zustände der Automatentheorie sind mit binären Signalen und Bitmustern darzustellen. Der schaltungstechnisch aufzubauende Zustandsautomat habe

- n Eingangssignale $e_1...e_n$,
- m Ausgangssignale $a_1...a_m$,
- p Zustandssignale $z_1...z_p$.

Tabelle 5.1 gibt an, wie die Anzahlen n, m und p des Zustandsautomaten mit den Mächtigkeiten der Mengen **E**, **A** und **Z** des abstrakten Automaten grundsätzlich zusammenhängen.

Zustandsautomat		**Abstrakter Automat**	
n Eingangssignale	$e_1...e_n$	Maximal 2^n Eingangsbelegungen	$E_1, E_2,...$
m Ausgangssignale	$a_1...a_m$	Maximal 2^m Ausgangsbelegungen	$A_1, A_2,...$
p Zustandssignale	$z_1...z_p$	Maximal 2^p Zustände	$Z_1, Z_2,...$

Tabelle 5.1 Zustandsautomat und abstrakter Automat. Grundsätzliche Zusammenhänge.

Die Signalbelegungen des Zustandsautomaten können als Binärvektoren **e**, **a** und **z** aufgefasst werden:

$$\mathbf{e} = (e_1...e_n)$$

$$\mathbf{a} = (a_1...a_m)$$

$$\mathbf{z} = (z_1...z_p)$$

Damit lassen sich die Überführungs- und Ausgabefunktionen folgendermaßen angeben:

Die Überführungsfunktion:

$$\mathbf{z}\,(t+1) = f\,(\mathbf{z}\,(t), \mathbf{e}\,(t)) \text{ oder kurz } \mathbf{z}^1 = f\,(\mathbf{z}, \mathbf{e})$$

Im Einzelnen ergibt sich die Überführungsfunktion als ein Bündel Boolescher Funktionen $f_i\,(\mathbf{z}, \mathbf{e})$ mit i = 1...p:

$$\begin{aligned} z^1{}_1 &= f_1\,(z_1...z_p, e_1...e_n) \\ z^1{}_2 &= f_2\,(z_1...z_p, e_1...e_n) \\ &.... \\ z^1{}_p &= f_p\,(z_1...z_p, e_1...e_n) \end{aligned} \qquad (5.1)$$

Die Ausgabefunktion des Mealy-Zustandsautomaten:

$$\mathbf{a}\,(t) = g\,(\mathbf{z}\,(t), \mathbf{e}\,(t)) \text{ oder kurz } \mathbf{a} = g\,(\mathbf{z}, \mathbf{e})$$

Im Einzelnen ergibt sich die Ausgabefunktion als ein Bündel Boolescher Funktionen $g_i\,(\mathbf{z}, \mathbf{e})$ mit i = 1...m:

$$\begin{aligned} a_1 &= g_1\,(z_1...z_p, e_1...e_n) \\ a_2 &= g_2\,(z_1...z_p, e_1...e_n) \\ &.... \\ a_m &= g_m\,(z_1...z_p, e_1...e_n) \end{aligned} \qquad (5.2)$$

Die Ausgabefunktion des Moore-Zustandsautomaten:

$$\mathbf{a}\,(t) = g\,(\mathbf{z}\,(t)) \text{ oder kurz } \mathbf{a} = g\,(\mathbf{z})$$

Im Einzelnen ergibt sich die Ausgabefunktion als ein Bündel Boolescher Funktionen $g_i\,(\mathbf{z})$ mit i = 1...m:

$$a_1 = g_1 (z_1...z_p) \qquad (5.3)$$
$$a_2 = g_2 (z_1...z_p)$$
$$....$$
$$a_m = g_m (z_1...z_p)$$

Jedes Bitmuster der Signale $e_1...e_n$, $a_1...a_m$ und $z_1...z_p$ ist im Grunde eine Codierung, die jeweils einem Element aus **E**, **A** oder **Z** entspricht. Zumeist ist die Codierung der Ein- und Ausgänge im Rahmen der Entwurfsaufgabe vorgegeben (von der Außenwelt gelieferte Eingangsbelegungen und – als Antwort darauf – zu erregende Ausgangssignale). Die Codierung der Zustände hingegen ist typischerweise frei wählbar (Abschnitt 5.3), es sei denn, die Zustandsbelegung gilt zugleich als zu liefernde Ausgabe (Medwedjew-Automat).

Die Überführungs- und Ausgabefunktionen (5.1), (5.2), (5.3) werden mit Funktionszuordnern dargestellt, die als kombinatorische Schaltnetze oder als adressierbare Speicher realisiert werden. Der typische adressierbare Speicher als Funktionszuordner ist ein Festwertspeicher (ROM), der mit den Parametern der jeweiligen Funktion adressiert wird und der an seinen Datenausgängen die Funktionswerte liefert (Tabelle 5.2).

Funktion	Eingänge (Adresse)	Ausgänge (Daten)	Speicherkapazität
Überführungsfunktion (5.1)	p + n	p	$2^{p+n} \bullet p$
Ausgabefunktion Mealy (5.2)	p + n	m	$2^{p+n} \bullet m$
Ausgabefunktion Moore (5.3)	p	m	$2^{p} \bullet m$

Tabelle 5.2 Grundsätzliche Anforderungen an die Funktionszuordner.

Die Abb. 5.11 und 5.12 zeigen die einfachsten Grundschaltungen von Mealy- und Moore-Automaten.

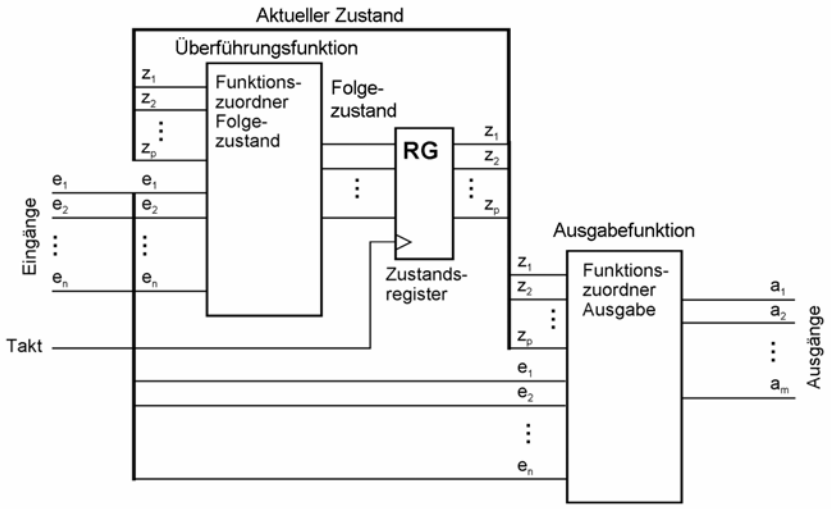

Abb. 5.6 Grundschaltung eines Mealy-Zustandsautomaten.

Dieser Mealy-Automat hat einen rein kombinatorischen Ausgangszuordner. Änderungen an den Eingängen können so noch im aktuellen Taktzyklus an den Ausgängen wirksam werden. Gelegentlich wird dies als besonderer Vorteil des Mealy-Automaten hervorgehoben. Manchmal ist dieses Verhalten tatsächlich erwünscht, manchmal nicht (das Schalten der Eingangssignale kann Störimpulse (Spikes, Glitches) hervorrufen).

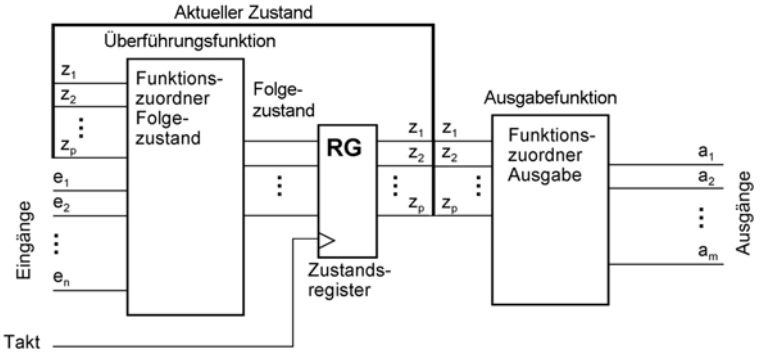

Abb. 5.7 Grundschaltung eines Moore-Zustandsautomaten.

Eingangsabtastung

Gemäß ist mit jedem Takt der zuvor ermittelte neue Zustand (Folgezustand) einzustellen und die Eingangsbelegung abzutasten. Um zu gewährleisten, dass sich der Automat als Abtastsystem verhält – also in jedem Taktzyklus nur einmal Information aus der Umwelt entnimmt – ist ein Eingangsregister vorzuschalten. Dass die Eingangsbelegung in jedem Taktzyklus nur einmal abgetastet wird, ist vor allem beim Mealy-Automaten von Bedeutung.

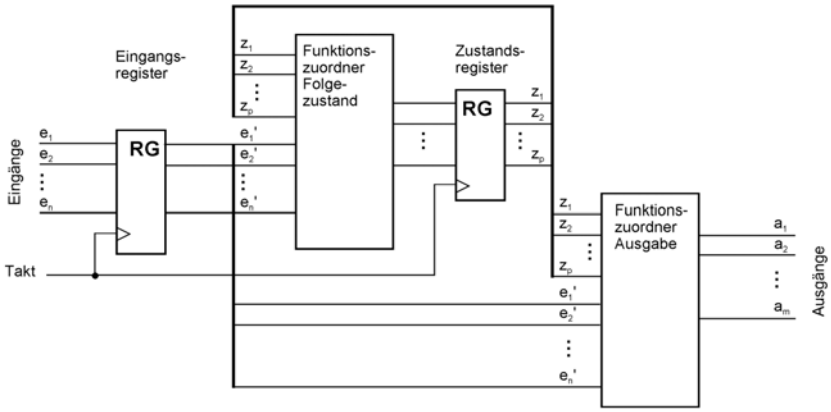

Abb. 5.8 Mealy-Zustandsautomat mit Eingangsabtastung. $e_1'...e_n'$ sind die abgetasteten (synchronisierten) Eingangssignale.

Moore-Zustandsautomat mit zeitverschobener Ausgabefunktion

Wenn Zustand und Eingangsbelegung den Folgezustand ergeben, so ergeben sie auch die Ausgangsbelegung im folgenden Taktzyklus.

Aus **a** (t) = g (**z** (t)) folgt **a** (t+1) = g (**z** (t+1)). Da z (t+1) = f (z(t), e(t)), kann man die Ausgabefunktion g (**z** (t+1)) auch in Abhängigkeit vom aktuellen Zustand und von der Eingangsbelegung angeben:

$$\mathbf{a}\,(t+1) = g\,(\mathbf{z}\,(t), \mathbf{e}\,(t)) \qquad (5.4)$$

Gemäß (5.4) wird die Ausgangsbelegung gleichzeitig mit dem Folgezustand bestimmt. Das erfordert einen aufwendigeren Funktionszuordner (mit p + n Eingängen, also wie beim Mealy-Automaten). Für den Durchlauf steht aber nahezu die gesamte Taktperiode zur Verfügung, und die Ausgangsbelegung erscheint gleichzeitig mit dem Folgezustand. Die Grundschaltung entspricht der des Mealy-Automaten. Ob man die Anordnung von Abb. 5.9 als Moore-Automaten mit zeitverschobener Ausgabefunktion oder als äquivalenten Mealy-Automaten bezeichnet, ist Ansichtssache. Von einem Moore-Automaten kann man dann sprechen, wenn jedem Folgezustand genau eine einzige Ausgangsbelegung zugeordnet ist (gleich auf welchem Wege der Zustand erreicht wurde).

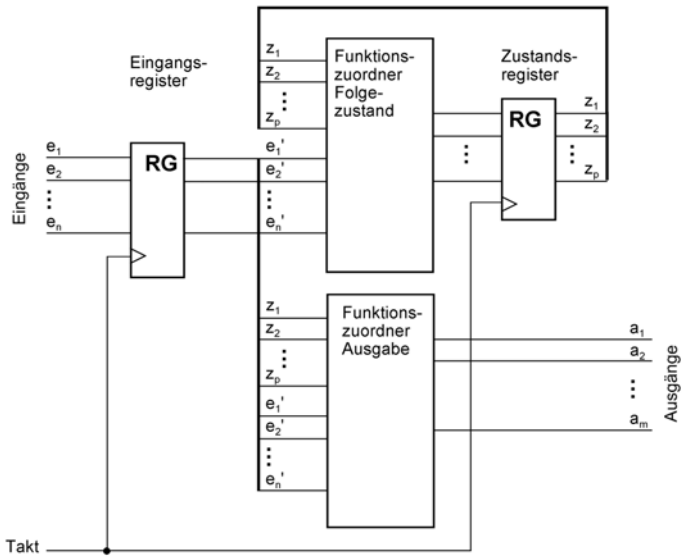

Abb. 5.9 Ein Moore-Zustandsautomat, der die Ausgangsbelegung gleichzeitig mit dem Folgezustand bestimmt (zeitverschobene Ausgabefunktion).

Kombinationslösungen

Ob ein Ausgangssignal ein Mealy-Verhalten oder ein Moore-Verhalten aufweist, wird lediglich durch die Booleschen Gleichungen der Ausgabefunktion bestimmt. Man kann somit verschiedene Verhaltensweisen in in einer einzigen Schaltung implementieren.

Vollsynchrone Zustandsautomaten

Bei vollsynchroner Arbeitsweise werden auch die Ausgangssignale über Register geführt. Damit verzögert sich die Ausgangsbelegung um einen Taktzyklus. Sie ist aber frei von Glitches, wie sie beim Durchlaufen von kombinatorischen Netzwerken oder beim Adressieren von Speichern auftreten können.

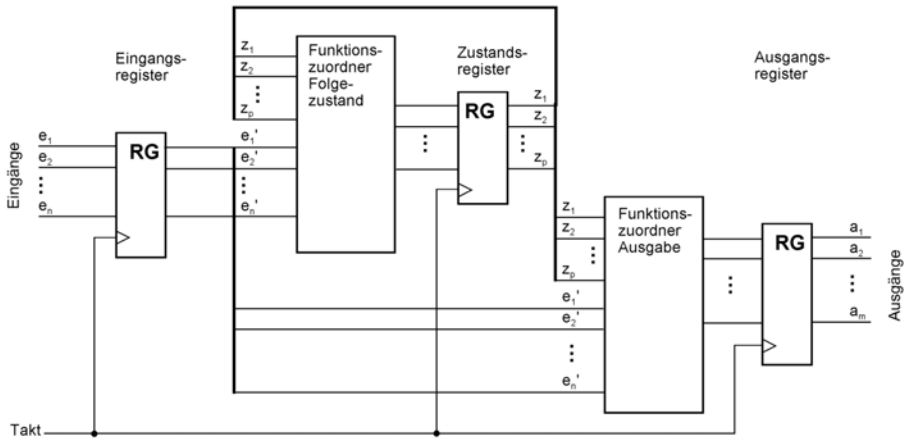

Abb. 5.10 Ein vollsynchroner Mealy-Zustandsautomat. Moore-Automaten können ebenso ausgelegt werden.

Asynchrone Automaten

Asynchrone Automaten haben keine von außen zugeführten Taktsignale. Der aktuelle Zustand wird über Rückführungen gehalten (die ausgangsseitigen Zustandssignale Z_A sind auf Zustandseingänge Z_R zurückgeführt;). Der Zustand ist dann stabil, wenn die aus der Eingangsbelegung E und den zurückgeführten Zustandssignalen Z_A (aktueller Zustand) gebildeten ausgangsseitigen Zustandssignale Z_F (Folgezustand) die gleiche Belegung aufweisen wie die zurückgeführten Zustandssignale Z_A (Folgezustand = aktueller Zustand):

$$f(E, Z_A) = Z_A$$

Der Zustand kann sich nur dann ändern, wenn sich die Belegung der Eingänge ändert. Damit sich der nächste stabile Zustand einstellen kann, sind in den Rückführungen ggf. Verzögerungsstufen vorzusehen.

Der asynchrone Automat hat in der Theorie eine viel größere Bedeutung als in der Praxis. Die Entwurfsschwierigkeiten sind beachtlich. Im Hinblick auf die modernen programmierbaren Schaltkreise ist von solchen Auslegungen grundsätzlich abzuraten. Eine gewisse Ausnahme bilden Interfacesteuerschaltungen für Signalspiele mit gegenseitiger Verriegelung (Interlocked Handshaking). Asynchrone Entwürfe beruhen zumeist auf RS-Latches mit mehreren Setz- und Rücksetzbedingungen sowie auf Flipflops, deren Eingänge trickreich ausgenutzt werden. Bei derartigen Entwürfen verlässt man sich typischerweise darauf, dass die jeweilige Gegenstelle am Interface (z. B. der Hostadapter) stets korrekte Signalspiele anbietet. Die Schaltfolge der Interfacesignale ersetzt dann praktisch den zentralen Takt. Die Vorteile: man spart die Takterzeugung ein, und es gibt auch keine Probleme mit der Synchronisation. Beides ist aber heutzutage kein wirklich bedeutsamer Kostenfaktor (so ist die Kostenersparnis zumeist schon dann hinfällig, wenn die asynchrone Lösung den Einsatz von Verzögerungsstufen erfordert).

5.3 Beschreibungsmittel

Zustandsdiagramm (Zustandsgraph)
Jeder Zustand wird durch einen Knoten (Node) dargestellt, die Zustandsübergänge werden durch gerichtete Verbindungen (Kanten; Edges) zwischen den Knoten angegeben. Typischerweise ist an jeder Verbindung (Kante) die Bedingung angetragen, die den jeweiligen Zustandswechsel auslöst. Das sind entweder einzelne Eingangsbelegungen oder Verknüpfungen von Eingangssignalen. Ist keine Bedingung eingezeichnet, so wird der Übergang im jeweils nächsten Taktzyklus zwangsläufig ausgeführt. Eine Rückführung auf denselben Knoten bedeutet, dass der aktuelle Zustand so lange beibehalten wird, wie keine Bedingung wirksam ist, um ihn zu verlassen. Bei Mealy-Automaten wird die Ausgabe an den auswärtsführenden Kanten vermerkt, beim Moore-Automaten in den Knoten. Oftmals stellt man aber nur die Zustände und die Übergänge zwischen ihnen dar (Medwedjew-Automat). Es gibt sowohl symbolische als auch formalisierte Darstellungen, z. B. die Zustandsgraphen der Entwicklungsumgebungen (vgl. Abb. 5.4a) und die State Charts der Modellierungssprache UML. Zumeist ist das Zustandsdiagramm aber eine graphisch mehr oder weniger ausgeschmückte Illustration.

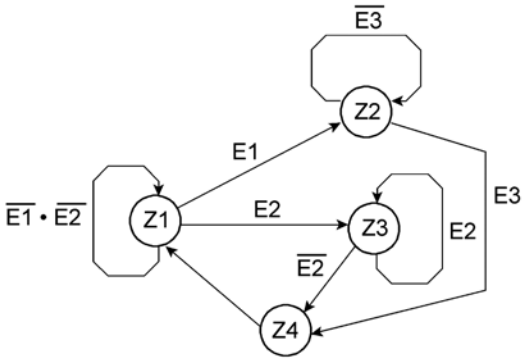

Abb. 5.11 Dieses Diagramm zeigt nur die Zustände und die Übergänge zwischen ihnen (Medwedjew-Automat).

Das Halten des aktuellen Zustandes
Das ist gelegentlich eine Quelle von Missverständnissen und Fehlern. Wichtig: ein korrekter Zustandsgraph muss eindeutig angeben, unter welchen Bedingungen der aktuelle Zustand verlassen wird. Ist keine Bedingung angegeben, so bedeutet dies, dass er nur einenTaktzyklus lang gehalten und dann unbedingt verlassen wird (vgl. Z4 in Abb. 5.11). Sind Bedingungen (Eingangsbelegungen) angegeben, so ist zu fragen, was geschieht, wenn keine der angegebenen Bedingungen eintrifft. In diesem Fall muss der aktuelle Zustand gehalten werden (Rückführungsschleife; vgl. Z1, Z2 und Z3 in Abb. 5.11). Manche Entwicklungssysteme können solche Rückführungen automatisch einfügen (Guaranteed Coverage). Ist diese Funktion aktiviert, wird jeder Zustand solange gehalten, bis eine Bedingung wirksam wird, die zum Verlassen des Zustandes führt (Übergangsbedingung).

Die Rückführungsbedingung entspricht der NOR-Verknüpfung aller Bedingungen, die zum Verlassen des Zustandes führen (vgl. Z1 in Abb. 5.11; $\overline{E1} \cdot \overline{E2} = \overline{E1 \vee E2}$). In manchen Diagrammen wird vorausgesetzt, dass es sich von selbst versteht (es ist dann nur die Schleife dargestellt, ohne Angaben von Bedingungen), in manchen hat man es jedoch schlicht und einfach vergessen.

Bedingungen, die Übergänge zu unterschiedlichen Zuständen veranlassen, dürfen nicht gleichzeitig erfüllt sein (vgl. Z1 in Abb. 5.11; es muss gelten $E1 \vee E2 = 0$ (disjunkte Bedingungen)). Sollte diese Forderung nicht erfüllt sein, wäre ggf. nachzuhelfen, z. B. durch Festlegen von Prioritäten. Beispiel (vgl. Z1 in Abb. 5.11): wenn E1 und E2 gleichzeitig wirksam werden, wird Zustand Z2 eingenommen (Vorrang von E1). Die Bedingung zum Übergang von Z1 nach Z2 wäre dann $E2 \cdot \overline{E1}$.

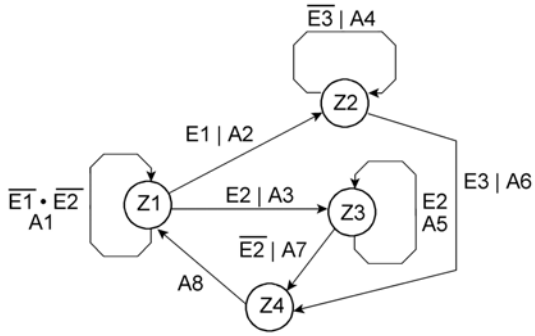

Abb. 5.12 Zustandsdiagramm eines Mealy-Automaten. Ablesebeispiel: Tritt in Zustand Z1 die Eingangsbelegung E1 auf, so wird die Ausgangsbelegung A2 wirksam, und im folgenden Taktzyklus wird der Zustand Z2 eingenommen.

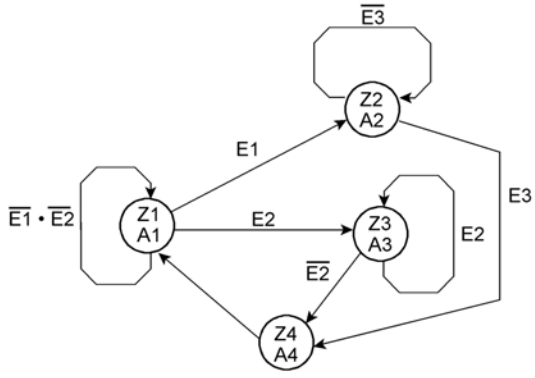

Abb. 5.13 Zustandsdiagramm eines Moore-Automaten. Ablesebespiel: Tritt in Zustand Z1 die Eingangsbelegung E1 auf, so wird im folgenden Taktzyklus der Zustand Z2 eingenommen. In diesem Zustand wird die Ausgangsbelegung A2 wirksam.

Bei gleichem Zustandsgraphen kann der Mealy-Automat mehr Ausgangsbelegungen haben als der Moore-Automat (Mealy: eine Ausgangsbelegung je Zustand und Eingangsbelegung, Moore: eine Ausgangsbelegung je Zustand). Um einen Moore-Automaten mit dem

Ausgangsverhalten des Mealy-Automaten von Abb. 5.12 zu bauen (äquivalenter Moore-Automat), muss der Automat von Abb. 5.13 um vier weitere Zustände ergänzt werden.

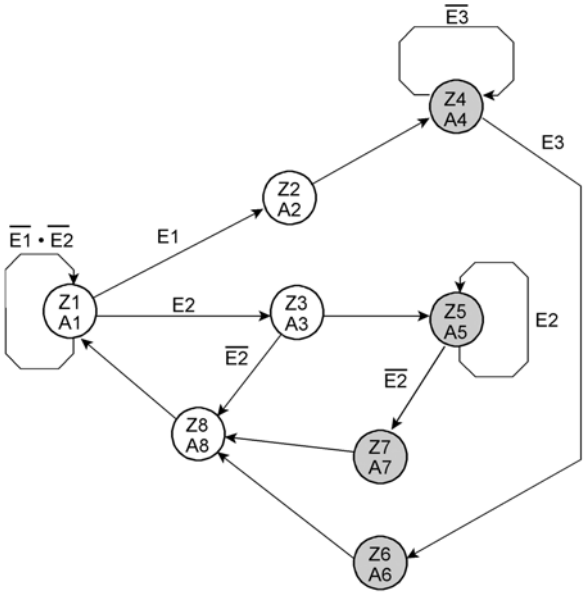

Abb. 5.14 Ein zum Mealy-Automaten von Abb. 5.17 äquivalenter Moore-Automat. Da man für jede Ausgangsbelegung einen Zustand braucht, müssen vier Zustände hinzugefügt werden. Die Bezeichnungen der Zustände wurden an die Bezeichnungen der Ausgaben angeglichen.

5.4 Zustandscodierung

Wenn man n Zustände vorsehen muss – wie viele Flipflops werden dazu benötigt und wie ordnet man deren Belegung den einzelnen Zuständen zu? Dies ist das Problem der Zustandscodierung. Grundsätzlich gibt es eine unabsehbare Vielfalt von Codierungsmöglichkeiten (auch sehr trickreiche), aber nur wenige haben in der Praxis weite Verbreitung gefunden.

Gibt es eine optimale Zustandscodierung?
In der Praxis geht es um geringste Kosten im Rahmen der gegebenen Technologie. Akademische Minimalkriterien (etwa aus der Schaltalgebra und Automatentheorie) sind dabei nur bedingt hilfreich (geringste Anzahl an Flipflops, geringster Aufwand für die Funktionszuordner). Bei der Wahl der Zustandscodierung sind vor allem folgende Gesichtspunkte von Bedeutung:

- Die Unterbringung in einem bestimmten Schaltkreis.
- Die Schalt- oder Taktzykluszeiten.
- Die vor- und nachgeschalteten Einrichtungen. Der Zustandsautomat sollte mit angemessenem Koppelaufwand in die umgebenden Schaltungen eingefügt werden können. Manchmal sind besondere Anforderungen zu erfüllen. Beispiel: glitchfreies Decodieren von Signalbelegungen.

- Störstrahlung und Stromsparen. Beide Anforderungen laufen darauf hinaus, die Anzahl der Schaltvorgänge zu verringern, auch wenn dies womöglich etwas mehr Schaltungsaufwand kostet.

Zustand	1 aus n (OHE)	Binär	Gray	Johnson
Z0	0000 0001	000	000	000
Z1	0000 0010	001	001	001
Z2	0000 0100	010	011	011
Z3	0000 1000	011	010	111
Z4	0001 0000	100	110	110
Z5	0010 0000	101	111	100
Z6	0100 0000	110	101	–
Z7	1000 0000	111	100	–
Anzahl der Bits für n Zustände	n	ld n	ld n	$^{n}/_{2}$

Tabelle 5.3 Typische Zustandscodierungen anhand von Beispielen. OHE: acht Zustände mit acht Flipflops; binär und Gray: acht Zustände mit drei Flipflops, Johnson: sechs Zustände mit drei Flipflops.

1-aus-n-Codierung (OHE)

Jeder Zustand wird durch ein Flipflop repräsentiert. Man braucht n Flipflops für n Zustände, wobei jeweils nur ein Flipflop aktiviert ist (One-Hot Encoding OHE). Das ist allem Anschein nach keineswegs ein Minimum, sondern eher ein Maximum an Aufwand. Dem ist aber nicht so. Offensichtlich gibt es Grenzen der Durchführbarkeit (tausende oder Millionen Zustände). Für geringere Anzahlen (Richtwert: bis zu einigen zehn Zuständen) hat das Prinzip aber einige Vorteile:

- Einfaches Entwerfen.
- Geringer Aufwand für die kombinatorischen Netzwerke.
- Wenige Schaltvorgänge (bei jedem Zustandswechsel schalten nur zwei Flipflops).
- Von vornherein ein Signal je Zustand (Decodieren nicht erforderlich),
- Die Zustandssignale werden glitchfrei gebildet; sie können somit als Taktimpulse genutzt werden.
- Das Umcodieren von 1-aus-n in beliebige andere Codes ist einfach.
- Die Funktionszuordner sind einfach und haben eine geringe Schaltungstiefe.
- Das Prinzip kommt den heutigen FPGA-Architekturen entgegen. Jede Zelle hat einen universellen Funktionszuordner für die Kombinatorik und ein Flipflop. Oftmals passt die vorzuschaltende Kombinatorik ohne weiteres in die Zelle, die das Flipflop enthält. Sparen von Flipflops bringt nichts – es sind schließlich alle bezahlt.

Binärcodierung
n Zustände werden mit ld n Flipflops[4)] binär codiert. Diese Anzahl ist das unumgängliche Minimum. Auch für sehr große Werte von n ist sie so gering, dass sich im Grunde beliebig viele Zustände codieren lassen. Diese Codierung hat aber auch grundsätzliche Nachteile:

- Je komplexer die Übergänge zwischen den Zuständen, desto schwieriger das Entwerfen, desto aufwendiger die kombinatorischen Netzwerke (die Schaltungen sind nur dann einfach, wenn auch die Zustandsübergänge einfach sind, z. B. beim Zählen).
- Die Bitmuster lassen sich nicht glitchfrei decodieren.
- Es können viele Schaltvorgänge auftreten (u. a. kann es vorkommen, dass alle Flipflops gleichzeitig schalten).

Optimale binäre Codierungen?
Das Problem: den einzelnen Zuständen Binärzahlen so zuzuordnen, dass insgesamt der Aufwand für die Funktionszuordner minimal wird. Dafür gibt es keine exakte Lösung[5)]. Manche Entwicklungssysteme begnügen sich mit dem Abprüfen einiger plausibler Gesichtspunkte (indem beispielsweise die Bitmuster so zugeordnet werden, dass sich beim Übergang von einem Zustand zum anderen möglichst wenige Schaltvorgänge ergeben). Sind die Bitmuster von Hand zuzuordnen, so genügt oftmals das einfache Durchummerieren von Null an. Werden adressierbare Speicher als Funktionszuordner eingesetzt, spielt die Kompliziertheit der kombinatorischen Verknüpfungen und damit die Codierung keine Rolle.

Gemischte Codierung
Manchmal sind bestimmte Codierungen in der Schaltungspraxis offensichtlich nicht anwendbar. Einen 20-Bit-Zähler (über eine Million Zustände!) wird man kaum mit 1-aus-n-Codierung verwirklichen. So liegt es nahe, Zustände, die durch Zählprozesse aufeinanderfolgen, binär zu codieren und solche, zwischen denen kompliziertere Übergänge vorgesehen sind, im 1-aus-n-Code.

Gray- und Johnson-Codes
Die Bitmuster werden so zugeordnet, dass sich beim Übergang von einem Zustand zum anderen nur die Belegung einer Bitposition ändert. Die Vorteile:

- minimale Anzahl an Schaltvorgängen beim Zustandswechsel,
- glitchfrei zu decodieren.

In strenger Form (minimale Anzahl an Bits, Änderung jeweils nur in einer Bitposition) sind diese Arten der Codierung allerdings nur nutzbar, wenn jeder Zustand nur einen einzigen Nachfolger. Das sind im Grunde die Zustandsübergänge eines Vorwärts-Rückwärts-Zählers mit Wartezuständen. Zulässig sind: (1) Übergang zum nächsten Zustand (Vorwärtszählen),

4: Genauer: ceil (ld n) (= auf die nächste ganze Zahl aufrunden).

5: Obwohl sich die Forschung über Jahrzehnte hinweg damit beschäftigt hat.

(2) Rückkehr zum vorhergehenden Zustand (Rückwärtszählen), (3) Verbleiben im aktuellen Zustand (Warten). Um mit diesen Codes andere Zustandsübergänge zu implementieren, sind Kompromisslösungen zu finden:

- Es wird in Kauf genommen, dass sich bei manchen Zustandsübergängen mehr als ein Bit ändert.
- Es werden Zwischenzustände eingeführt. Das erfordert mehr Bits (also Flipflops) und mehr Taktzyklen.

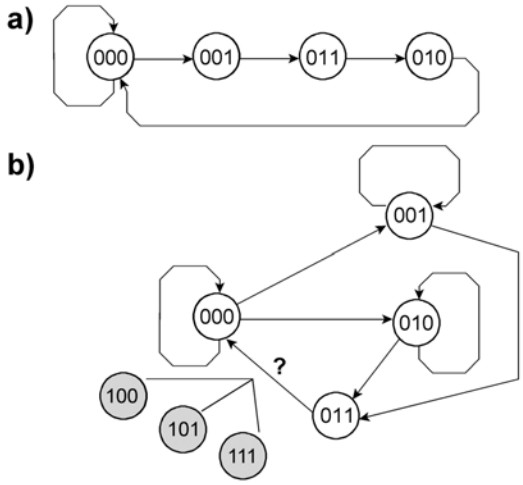

Abb. 5.15 Zustandscodierung gemäß Gray-Code. a) jedem Zustand lässt sich ohne weiteres ein Codewort des Gray-Codes zuordnen. b) in diesem Beispiel gelingt es nicht. Ein Ausweg: es werden Zwischenzustände eingefügt, die nacheinander durchlaufen werden.

Der Gray-Code ist im Grunde eine Sonderform der binären Codierung (ld n Bits für n Zustände). Der Johnson-Code erfordert $^{n}/_{2}$ Flipflops, die kombinatorischen Verknüpfungen sind aber typischerweise einfacher.

Zustandscodierung = Ausgangsbelegung

Die Zustandscodierung wird so gewählt, dass Zustandssignale direkt als Ausgangssignale verwendet werden können (Medwedjew-Automat). Dieser Ansatz führt oftmals zu günstigen Lösungen. Es gilt:

- Wenn jedem Zustand eine eigene Ausgangsbelegung zugeordnet ist, sind soviele Flipflops erforderlich, wie Ausgangssignale zu liefern sind.
- Sind mehreren Zuständen gleiche Ausgangsbelegungen zugeordnet, so sind zusätzliche Flipflops erforderlich, um die Zustände voneinander unterscheiden zu können.

Faustregeln:

- Diskreter Entwurf (SSI/MSI): je nach Anzahl der Zustände und Kompliziertheit der Zustandsübergänge binäre oder OHE-Codierung oder Tricklösung. Aus der Sicht des allgemeinen Kriteriums "so wenig Schaltkreisgehäuse wie möglich" (Minimum Package Count) dürfte zumeist OHE vorzuziehen sein, weil sich dabei einfache Verknüpfungen mit wenigen Eingängen ergeben, die sich mit den typischen Gatterschaltkreisen (Einzelgatter usw.) kostengünstig aufbauen lassen.
- ROM- oder RAM-Zuordner: unbedingt binär. Jedes eingesparte Zustandssignal verringert die Anforderungen an die Speicherkapazität auf die Hälfte! Die Kompliziertheit der kombinatorischen Verknüpfungen spielt keine Rolle.
- CPLDs: ähnlich SSI/MSI. CPLD-Architektur (Zellen, Produktterme, globale Verbindungsressourcen) ansehen und überlegen, was am besten passen könnte. Umfangreiche UND-ODER-Strukturen sprechen eher für binäre Codierung. Es kann aber sein, dass dies die Verbindungsressourcen zustopft. Als Alternative kann man damit beginnen, die Flipflops auszunutzen (Prinzip: wenn es genügend Flipflops gibt, dann OHE verwenden).
- FPGAs und Gate Arrays: die Grundsatzentscheidung: adressierbare Speicher oder Schaltnetze? Geht es um nicht allzu viele Zustände (Richtwert: einige zehn) und ist die Überführungsfunktion nicht allzu kompliziert, so ist das Schaltnetz meist die bessere Wahl (es braucht weniger Ressourcen, lässt die RAM-Blöcke frei und kann typischerweise mit höheren Taktfrequenzen betrieben werden). Hierfür OHE bevorzugen, da die Zellen ohnehin Flipflops enthalten. Bei OHE werden die kombinatorischen Verknüpfungen oftmals so einfach, dass sie mit dem Funktionszuordner jener Zelle erledigt werden können, die das Flipflop enthält. Wichtig: minimale Zellenzahl in den Grenzen der ausführbaren Verbindungen zwischen den Zellen, nicht aber minimale Flipflop- oder Gatteranzahl.

Unzulässige Zustände

Sie sollten eigentlich nicht vorkommen. Der Anfangszustand ist typischerweise durch Rücksetzen zu erzwingen. Ist damit zu rechnen, dass die Schaltung einen unzulässigen Zustand einnehmen kann, so sind die unzulässigen Bedingungen zu erkennen (Decodierung), und es ist ein Übergang in einen Fehlerzustand (oder in den Anfangszustand) zu veranlassen.

Zustandscodierung in Entwicklungssystemen

Typischerweise stehen mehrere Möglichkeiten zur Wahl:

- automatische oder manuelle Auswahl,
- vorrangige Entwurfsziele für die automatische Auswahl (hohe Geschwindigkeit oder geringer Aufwand),
- verschiedene übliche Codierungen, z. B. binär und 1 aus n (One-Hot oder One-Cold).

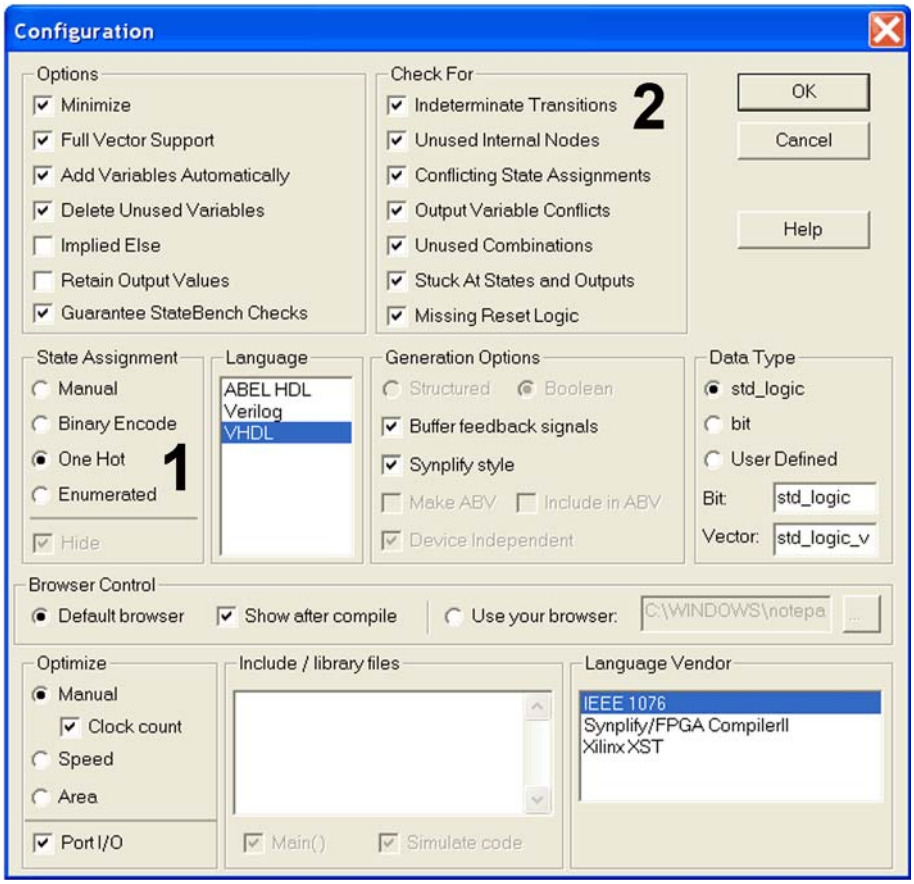

Abb. 5.16 Konfigurationssteuerung beim Entwerfen mit Zustandsdiagrammen ([S04]). 1 - Auswahl der Zustandscodierung; 2 - Auswahl der Korrektheits- und Plausibilitätsprüfungen.

OHE-Schaltungsentwurf

Geht es um vergleichsweise wenige Zustände, ist die 1-aus-n-Codierung (One-Hot Encoding OHE) oftmals die zweckmäßigste Lösung. Solche Zustandsautomaten können auch von Hand systematisch entworfen werden. Die Vorgehensweise wird im Folgenden für verschiedene Flipfloptypen anhand eines einzelnen Zustands erläutert.

Der hier in Rede stehende Zustand kann von anderen Zuständen aus erreicht (gesetzt) und in Richtung anderer Zustände verlassen werden. Ist er aktiv und sind keine Übergangsbedingungen in andere Zustände erfüllt, so wird er gehalten. Die Bedingungen des Setzens und Verlassens sind jeweils einzelne Signale oder Boolesche Variable, die aus dem Zustandsdiagramm abgelesen werden können. Signalkombinationen müssen ggf. durch Decodierung in Einzelsignale aufgelöst werden.

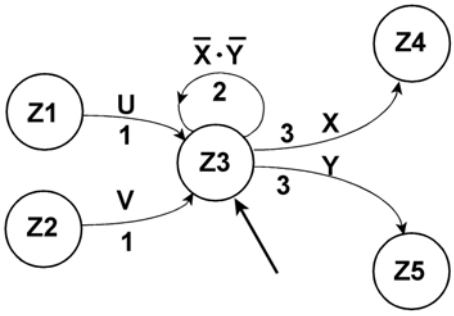

Abb. 5.17 Ausschnitt aus einem Zustandsdiagramm als Beispiel für den OHE-Schaltungsentwurf. Gesucht ist eine Flipflopschaltung, die den Zustand Z3 darstellt (Pfeil). 1 - Setzbedingungen; 2 - Haltebedingung; 3 - Übergangsbedingungen.

Anfangszustand
Der Anfangszustand ist nach dem Einschalten hart zu erzwingen. Hierzu ist das Flipflop des Anfangszustandes zu setzen. Alle anderen Flipflops sind zurückzusetzen.

Erreichen des betreffenden Zustandes (Setzbedingung)
Der Zustand wird erreicht, wenn der bisherige Zustand verlassen wird:

> Setzbedingung = bisheriger Zustand UND Übergangsbedingung in den betreffenden Zustand.

Gibt es mehrere Setzbedingungen, so sind sie disjunktiv zu verknüpfen.

Halten des Zustandes (Haltebedingung)
Wenn keine Übergangsbedingung zum Verlassen des Zustandes erfüllt ist, muss er beibehalten werden. Die Haltebedingung entspricht der NOR-Verknüpfung aller Übergangsbedingungen, die zum Verlassen des Zustandes führen:

> Haltebedingung = aktueller Zustand UND kein Übergang in einen anderen Zustand.

Verlassen des Zustandes (Übergangsbedingung)
Der Zustand wird verlassen, wenn eine entsprechende Übergangsbedingung wirksam wird. Die Übergangsbedingung (der Boolesche Ausdruck an der jeweiligen Kante des Zustandsdiagramms) ist zugleich Setzbedingung für den nachfolgenden Zustand.

> Verlassen des aktuellen Zustandes = aktueller Zustand UND Übergangsbedingung in einen anderen Zustand.

Gib es mehrere Übergangsbedingungen, so sind sie disjunktiv zu verknüpfen.

Die Übergangsbedingung eines jeweils vorhergehenden Zustandes ist zugleich eine Setzbedingung des jeweils nachfolgenden Zustandes; die UND-Verknüpfungen von Zustand und Eingangssignalen (Übergangsbedingungen) dienen gleichzeitig zum Verlassen (Ausschalten) des aktuellen und zum Setzen des nachfolgenden Zustandes.

Implementierung des Zustands mittels D-Flipflop
Das Flipflop ist zu setzen, wenn eine Setzbedingung erfüllt und der jeweils zu verlassende Zustand aktiv ist. Das Flipflop ist aktiv zu halten (Rückführung), wenn keine der Übergangsbedingungen erfüllt ist. Der Zustand wird verlassen, wenn die Haltebedingung inaktiv wird und hierdurch das Flipflop auf Null schaltet.

$$D_{Z3} = Z3_{setzen} \vee Z3_{halten}$$

$$Z3_{setzen} = Z1 \cdot U \vee Z2 \cdot V \qquad (5.10)$$

$$Z3_{halten} = Z3 \cdot \overline{X \vee Y} = Z3 \cdot \overline{X} \cdot \overline{Y}$$

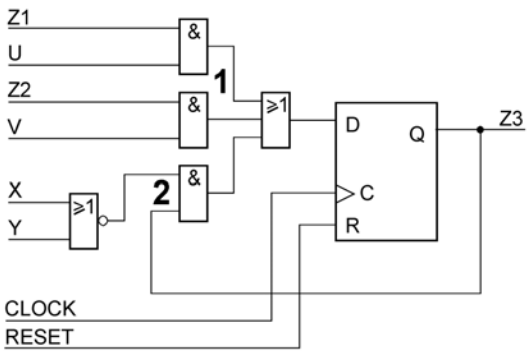

Abb. 5.18 Implementierung des Zustands mittels D-Flipflop. 1 - Setzen; 2 - Halten.

Implementierung des Zustands mittels T-Flipflop
Die Toggle-Funktion ist dann zu aktivieren, wenn der Zustand des Flipflop zu ändern ist. Es gibt zwei Fälle:

1. Setzen. Eine Zustandsänderung von 0 nach 1 ist auszulösen, wenn eine Setzbedingung erfüllt und der jeweils zu verlassende Zustand aktiv ist.

2. Löschen. Eine Zustandsänderung von 1 nach 0 , wenn der Zustand aktiv und eine der Übergangsbedingungen zum Verlassen des Zustandes erfüllt ist. Hierzu sind die Setzbedingungen der Folgezustände disjunktiv zu verknüpfen.

$$T_{Z3} = Z3_{setzen} \vee Z3_{verlassen}$$

$$T_{Z3} = Z1 \cdot U \vee Z2 \cdot V \vee Z3 \cdot X \vee Z3 \cdot Y \qquad (5.11)$$

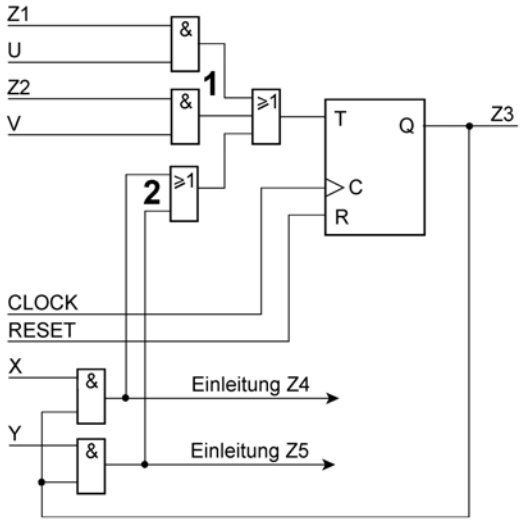

Abb. 5.19 Implementierung des Zustands mittels T-Flipflop. 1 - Setzen; 2 - Verlassen.

Implementierung des Zustands mittels JK-Flipflop

Das Flipflop ist zu setzen, wenn eine Setzbedingung erfüllt und der jeweils zu verlassende Zustand aktiv ist. Das Flipflop ist zurückzusetzen, wenn der Zustand aktiv und eine der Übergangsbedingungen zum Verlassen des Zustandes erfüllt ist.

$$J_{Z3} = Z3_{setzen} = Z1 \cdot U \vee Z2 \cdot V$$
$$K_{Z3} = Z3_{verlassen} = Z3 \cdot X \vee Z3 \cdot Y \qquad (5.12)$$

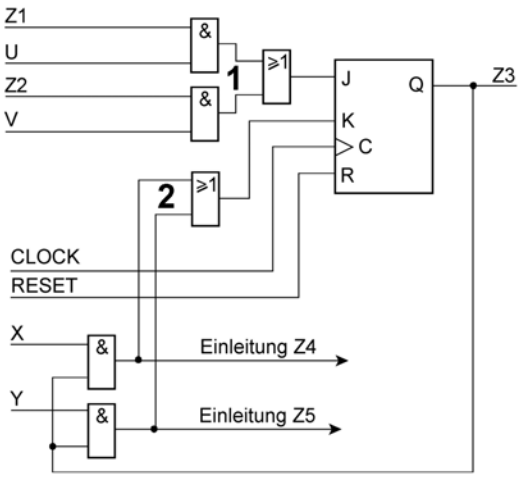

Abb. 5.20 Implementierung des Zustands mittels JK-Flipflop. 1 - Setzen; 2 - Verlassen.