Ergänzung zum Beitrag in FA 6/17, S. 535 ff.
„Spannungs- und Stromüberwachung für Schalt- und Linearregler“

■ Etwas Automatentheorie

Den Hintergrund bildet die sogenannte Automatentheorie, die englisch meist mit dem Begriff *Finite-State-Machine* [1] völlig anders bezeichnet wird. Der Kern der Theorie besteht darin, dass man formal jeden Vorgang und jeden Algorithmus als eine Ansammlung von Zuständen und Übergängen mit den dazugehörigen Bedingungen zum Wechsel zwischen den Zuständen darstellen kann. Das trifft z. B. beim Ritual des morgendlichen Aufstehen genauso zu, wie bei der Entwicklung einer Hard- oder Software.

Bei einer synchronen State-Machine ist der Wechsel zwischen den *States* (englisch für Zustände) nur zu einem festgegeben Zeitpunkt möglich. Sie eignen sich deshalb hervorragend für übersichtliche Steuerprogramme, wie z. B. die Spannungs- und Stromüberwachung. Links im Bild 1 findet man die Legende für das Zustandsdiagramm. Die Zustände jeder State-Machine werden mit Kreisen grafisch dargestellt. Sie sind mit Übergangspfeilen verbunden, an denen die Bedingung für den Wechsel und die zugehörige Ausgabe steht.

Die Eingabebedingung für den Zustandswechsel ist hier zur besseren Verdeutlichung in blauer Schrift und die zugehörige Ausgabe in roter Schrift dargestellt. Es ist durchaus möglich, dass nicht nur eine, sondern mehrere Bedingungen zu einem Wechsel in den gleichen Folgezustand auftreten. Das entspricht formal einer ODER-Verknüpfung, während eine UND-Verknüpfung bereits in jeder Bedingung, die aus mehr als einer Eingangsvariablen (bei der Spannungs- und Stromüberwachung aus vier) besteht, zusammen gefasst wird. Kommentare stehen im Diagramm neben einer gestrichelten Linie.

Für den Fall der Spannungs- und Stromüberwachung kann man das ganze Programm durch Zustände und Übergänge wie in Bild 1 rechts darstellen. Die Darstellung ist nur begrenzt mit einem Flussdiagramm vergleichbar, denn in einem Flussdiagramm findet man grafisch z. B. keine zeitlich feste Taktung des Ablaufs, sondern eine Beschreibung mit Anweisungen, Schleifen und Abfragen. Ein State-Diagramm enthält dagegen nur Bedingungen und daraus folgende Reaktionen, ohne zu beschreiben, wie die Bedingungen generiert werden. Die müssen quasi unabhängig davon „angeliefert“ werden. Im Beispiel der Spannungs- und Stromüberwachung ist das z. B. die Erzeugung der beiden Messwerte für die Ausgangsspannung U_A und den Laststrom I_A. Beide Werte werden im Programm im Hintergrund erzeugt (gemessen), ohne dass die State-Machine unmittelbar damit etwas zu tun hat! Ähnliches gilt für die Erzeugung und Ausgabe des aktuellen Blinkmusters. Diese Trennung entlastet die Entwicklung komplexer Programme ganz erheblich, weil alle Programmteile unabhängig voneinander entwickelt und getestet werden können.

Beim Blick auf das eigentliche Diagramm (rechts in Bild 1) sieht man auch Übergänge, die wieder auf den gleichen Zustand zurück führen. So ist z. B. der grün hinterlegte Zustand Z2 dafür zuständig die Ausgangsspannung und den Laststrom zu überwachen. Wenn die vorgegebenen Grenzwerte von U_A und I_A nicht überschritten werden, dann bleibt der Zustand Z2 natürlich ständig erhalten. Nur bei Überschreitung wird in den Zustand Z3 gewechselt und die entsprechenden Ausgangssignale verändert!

In einem Flussdiagramm müsste man auch beschreiben, was in der Zwischenzeit zwischen zwei Taktimpulsen getan werden soll. In einem State-Diagramm wird also tatsächlich nur auf die Änderungen der Eingangsgrößen reagiert, was nur wenig Rechenzeit benötigt. Bis zum nächsten Taktzeitpunkt kann der Prozessor daher etwas beliebig anderes tun und sich z. B. um die Digitalisierung und Mittelung der Messwerte oder die Ausgabe des aktuellen Blinkmusters kümmern. Die meiste Zeit (hier über 95 %) wartet er jedoch in einer Schleife auf das nächste Ereignis.

Der Vollständigkeit halber muss erwähnt werden, dass das Programm der Spannungs- und Stromüberwachung nicht ganz exakt der dargestellten Definition eines *Mealy*-Automaten entspricht. Es ist vielmehr eine Mischform mit einem sogenannten Moore-Automaten und wer den Assembler-Quelltext genau studiert, der wird auch die eine oder andere Erweiterung finden, die

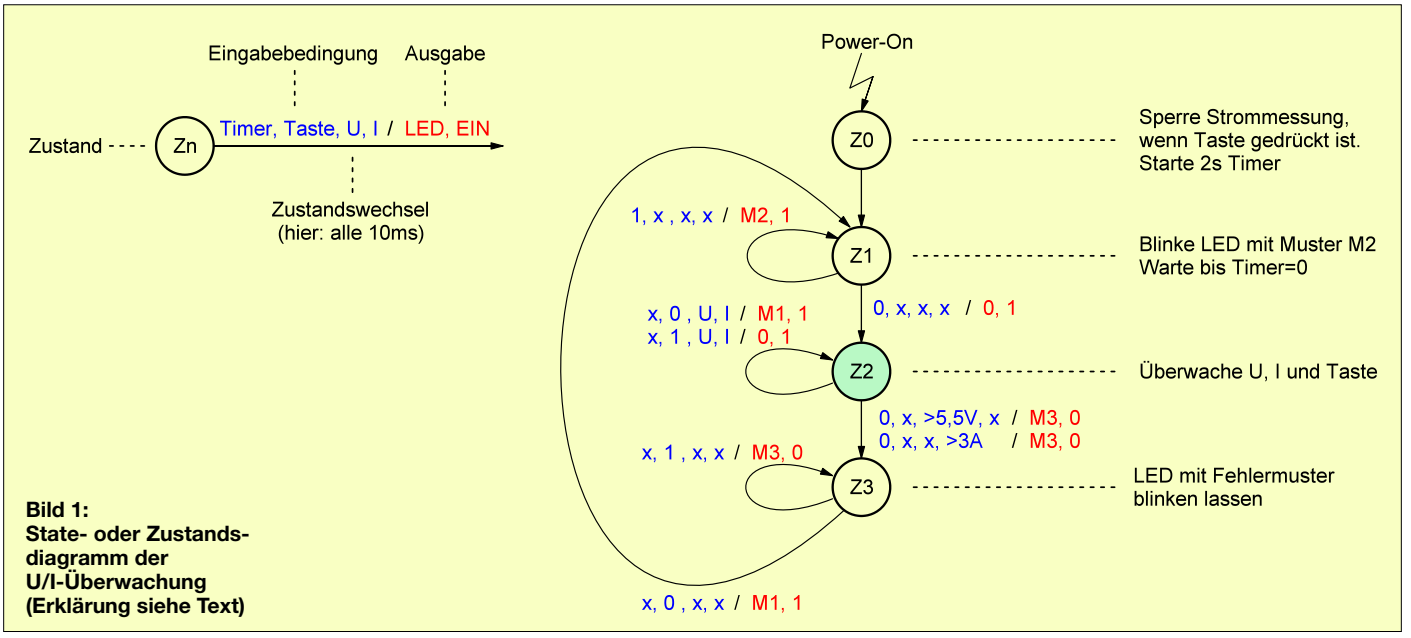

Bild 1:
State- oder Zustandsdiagramm der U/I-Überwachung (Erklärung siehe Text)

im Bild 1 für eine bessere Übersicht weg gelassen wurde. Am grundsätzlichen Verständnis ändert das jedoch nichts.

Beschreibung eines Zustandswechsels

An der bei Z3 links stehenden Eingangsbedingung x, 1 ,x , x kann man ablesen, dass nur der Tasteneingang einen Einfluss auf alles weitere hat. So lange die Taste nicht gedrückt wird steht am Eingang eine logische 1 (H-Pegel). Alle anderen Eingangsgrößen sind beliebig (mit x gekennzeichnet). Der Zustand Z3 kann also nur in Richtung Z1 verlassen werden, wenn die Taste gedrückt wird (Eingang logisch 0). Die Eingangsbedingung dazu muss dann x, 0 ,x , x lauten! In diesem Fall wechselt die Ausgabe des Blinkmusters von M3 nach M1 und an den Regler wird anstatt 0 wieder eine 1 ausgegeben (M1, 1), so dass er eingeschaltet wird. Bedingungen für Übergänge zu anderen Zuständen als Z1 gibt es in Z3 nicht. Sie können also auch nie auftreten! Was in Zustand Z0 und Z1 passiert, kann man sich nun zusammen mit den Kommentaren leicht selbst erarbeiten.

In der vorliegenden Steuerung beträgt die Zykluszeit 10 ms. Sie weiter zu reduzieren wäre prinzipiell möglich, sofern die zusätzlich im Hintergrund laufenden Routinen noch schnell genug sind. Im vorliegenden Steuerprogramm liegen die Grenzen der Zykluszeit bei der Messung der beiden Werte *U* und *I*, abwechselnd in jeder Millisekunde. Ein gesamter Messzyklus dauert also 2 ms. Eine noch schnellere Reaktion als 10 ms würde kaum etwas bringen, da dann auch die Störempfindlichkeit zunimmt. Bereits jetzt wird zur Unterdrückung nämlich jeder Spannungs- und Strommesswert gleitend mit dem letzten Wert gemittelt, um Störspitzen etwas abzuflachen. Tatsächlich werden in 10 ms also bereits fünf Messwerte pro Spannung und Strom verwendet und damit eine recht gute Störsicherheit erzeugt. Die Zykluszeit von 10 ms ist hier also durchaus angemessen und keinesfalls zu langsam gewählt. Normale Schmelzsicherungen reagieren mindestens um den Faktor 10 langsamer! In realen Prozesssteuerungen passt man die Zykluszeit daher immer dem zu steuernden Prozess an und nicht unbedingt irgend einer unüberlegten Ideologie der schnelleren Reaktionszeit.

Vorteile von synchronen Automaten

Der Entwurf und vor allem die Wartung und Erweiterung komplexer Programmsysteme wäre ohne solche strukturieren Darstellungen völlig unmöglich. Jeder Zustand kann bei einem Zustandsautomaten unabhängig von den anderen Zuständen als Unterprogramm und gleichzeitig von verschiedenen Programmierern codiert werden. Der Vorteil macht sich erst dann richtig bemerkbar, wenn ein Programm aus mehreren hundert Zuständen besteht, bringt dann aber einen erheblichen Zeitvorteil bei der Programmentwicklung.

Ein besonderes Merkmal synchroner Automaten (bzw. einer synchronen State-Machine) ist ihre definierbare Reaktionszeit. Sie kann nur so schnell reagieren, wie ihre eigene Takt- oder Zykluszeit das zulässt. Damit wird die Reaktionszeit zugleich auch von der Taktfrequenz des Controllers entkoppelt. Was zunächst nachteilig klingt, ist jedoch durchaus erwünscht, denn viele DSP-Algorithmen (DSP = *Digital Signal Processing*) benötigen unbedingt synchrone Muster für eine Bearbeitung. Die Anbindung an einen Automatentakt ermöglicht es auch, dass auf einem Prozessor quasi gleichzeitig verschiedene voneinander unabhängige Programme laufen können. Multi-Tasking-Systeme wie Linux haben daher im Kern auch eine State-Machine (den sog. Scheduler), die die Vergabe von Rechenzeit und anderen Systemresourcen geschickt organisiert.

Ein asynchroner Spaghetti-Code würde dagegen zu völlig unkontollierbaren Reaktionszeiten führen, die auch noch von der Rechenzeit der anderen Programme abhängt. So ein Verhalten wäre im Beispielprogramm zwar noch tolerierbar, für die meisten Prozesssteuerungen, z. B. einer Fertigungsstraße, jedoch nicht akzeptabel. Asynchrone State-Machines ohne Taktsteuerung gibt es auch, doch sie sind erheblich komplexer zu entwicklen und sollen daher hier nur der Vollständigkeit erwähnt werden. Sie finden überall da Anwendung, wo es auf äußerst zeitkritische Reaktionen (z. B. in der internen Ablaufsteuerung eines modernen Mikroprozessors) ankommt. Leider sind sie auch deutlich fehleranfälliger, da sie nicht nur zu einem vorgegebenen, fixen Zeitpunkt reagieren müssen und ihre Ausgangssignale ein gegebenenfalls störendes, zeitliches Jittern aufweisen können.

dc7gb@vfdb.org

Literatur

[1] Wikipedia: Finite-State-Machine. *https://en.wikipedia.org/wiki/Finite-state_machine*