

Beispiel zum Schaltungsentwurf mithilfe endlicher Automaten

Ein Zähler modulo 3 mit Reset

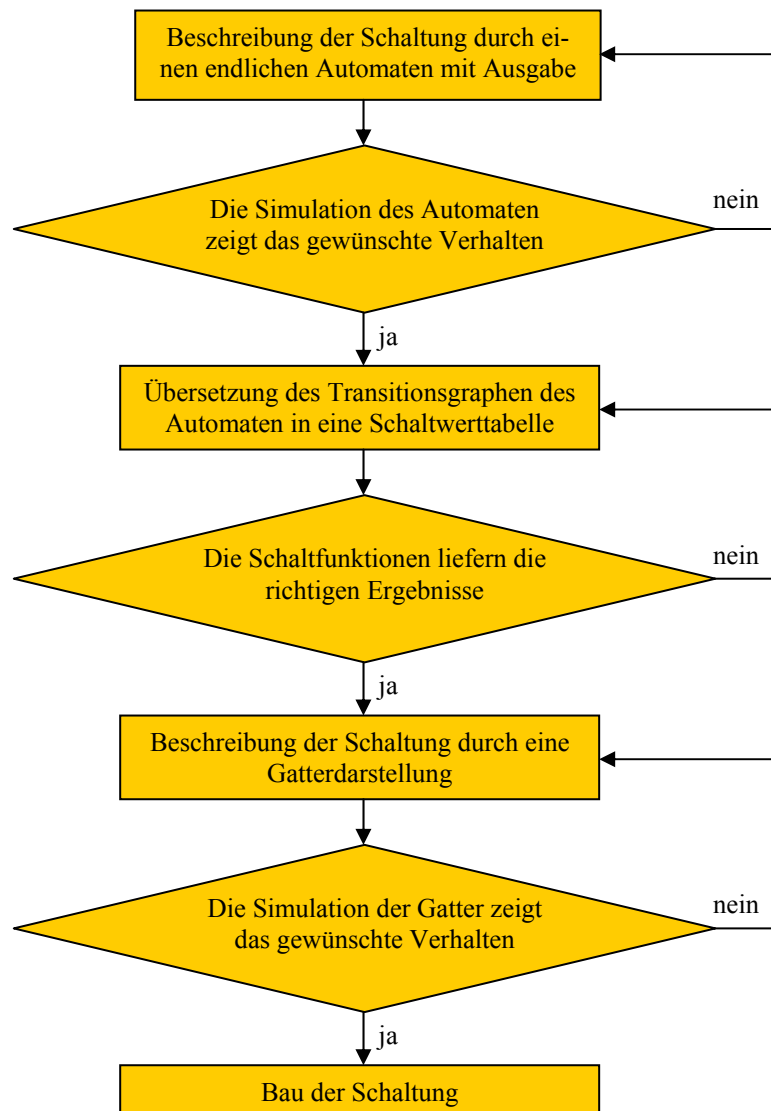
Inhalt:

1. Bezug zum Unterricht
2. Beschreibung durch einen endlichen Automaten
3. Überprüfung durch ein Programm
4. Aufstellen der Schwertabelle
5. Entnehmen der Schaltfunktionen
6. Überprüfung der Schaltfunktionen
7. Test im Digitalsimulator
8. Test mit HASI

1. Bezug zum Unterricht

Schaltwerke können in aus praktischen Gründen (z. B. Platz und Übersichtlichkeit) begrenztem Rahmen systematisch mithilfe von endlichen Automaten entwickelt werden. In diesem Sinne kann die technische Informatik in der Schule eine Vorstufe (oder ein Teilgebiet) der theoretischen Informatik darstellen. Aus unterrichtsorganisatorischen Gründen ist es sehr empfehlenswert, diesen Weg zu gehen, weil die Nutzung der und die Gewöhnung an Notationsformen der Automatentheorie einen Theoriekurs, der meist dicht vor dem Abitur liegt, effizient vorbereitet. So kann die theoretische Informatik auf einem der Sek. II angemessenen Niveau unterrichtet werden, ohne aus Zeitgründen in Anfangsproblemen stecken zu bleiben.

Systematischer Schaltungsentwurf bedeutet in diesem Kontext, dass Schaltwerke in unterschiedlicher Detailliertheit und unterschiedlichem Abstraktionsgrad auf unterschiedliche Modelle abgebildet werden. Vor dem Wechsel des Modells sollte jeweils (so weit wie möglich und sinnvoll) überprüft werden, ob das gerade entwickelte Modell seinen Anforderungen entspricht. Daraus ergibt sich eine Folge von Modellen und Tests, die endlich zu einer konkreten Schaltung führt, die (hoffentlich) den ursprünglich gestellten Anforderungen entspricht.

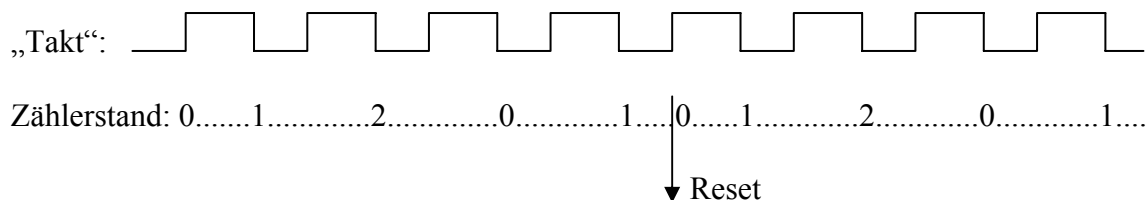


Eine solche klar gegliederte Organisation des Unterrichts ermöglicht es außerdem, je nach Zeitkontingent das Thema an unterschiedlichen Stellen sinnvoll zu beenden oder weiter zu vertiefen. Auf allen Ebenen ist aktive Schülerarbeit möglich.

Die so entwickelten Komponenten (Rechenschaltungen, Speicher, Zähler, ...) können dann in einem zweiten Anlauf als Bausteine benutzt werden, aus denen kompliziertere Schaltwerke oder sogar Modellcomputer „komponiert“ werden. Die hierfür erforderliche Arbeitsweise entspricht dann einem „technischen“ Umgang mit den Komponenten:

- Wir versuchen einerseits, Computer zu „verstehen“, indem wir sie teilweise nachbauen.
- Andererseits erwerben wir Wissen nicht unbedingt um unser Verständnis für die Welt zu erhöhen, sondern um „etwas zu tun“.

Als Beispiel für den systematischen Entwurf von Schaltungen wollen wir einen **Zähler** entwickeln, der **modulo 3** arbeitet und zusätzlich durch Eingabe des Zeichens **R** (Reset) jederzeit in den Anfangszustand gebracht werden kann. Der Zähler liest ansonsten das „normale“ Taktsignal, also eine Leitung, die periodisch zwischen den Werten 0 und 1 wechselt. Dabei zählt die Schaltung wiederholt von Null bis Zwei: 0-1-2-0-1-2-0-1-2-... Der Zähler ändert seine Anzeige, wenn die Eingabe von 1 auf 0 wechselt.



2. Beschreibung durch einen endlichen Automaten

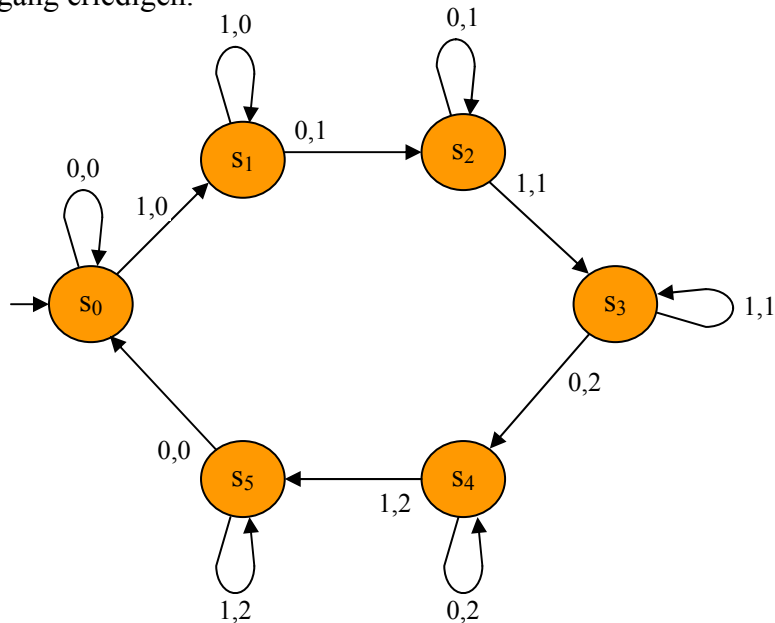
Zuerst bestimmen wir die Alphabete:

Eingabealphabet: $\mathbb{E} = \{0,1,R\}$

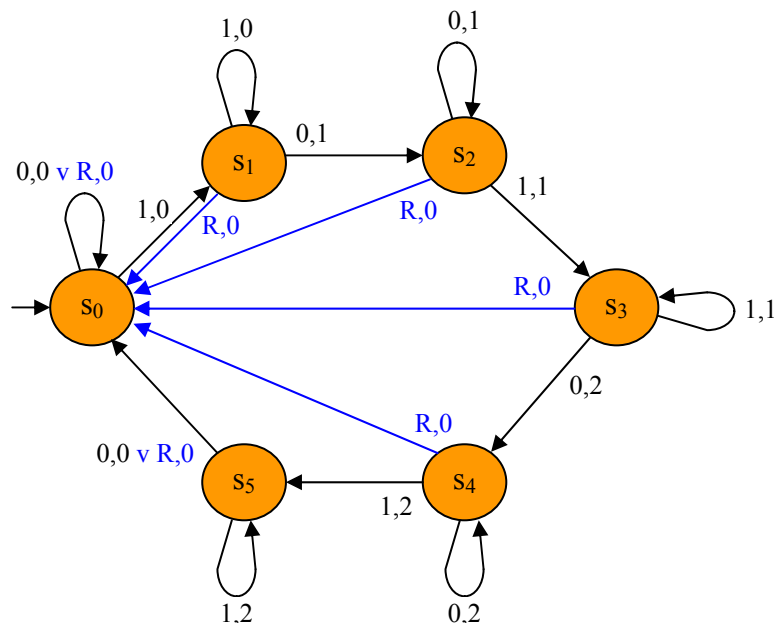
Ausgabealphabet: $\mathbb{A} = \{0,1,2\}$

Die Zustandsmenge ergibt sich nach Konstruktion des Transitionsgraphen.

Da wir die Wechsel der Taktleitung mitverfolgen müssen, wählen wir Zustände, die dieses für einen einfachen Durchgang erledigen.



Danach vervollständigen wir den Graphen, indem wir dafür sorgen, dass in jedem Zustand für jedes Eingabezeichen eine angemessene Reaktion definiert ist. Es fehlt noch die Reaktion auf das Zeichen R.

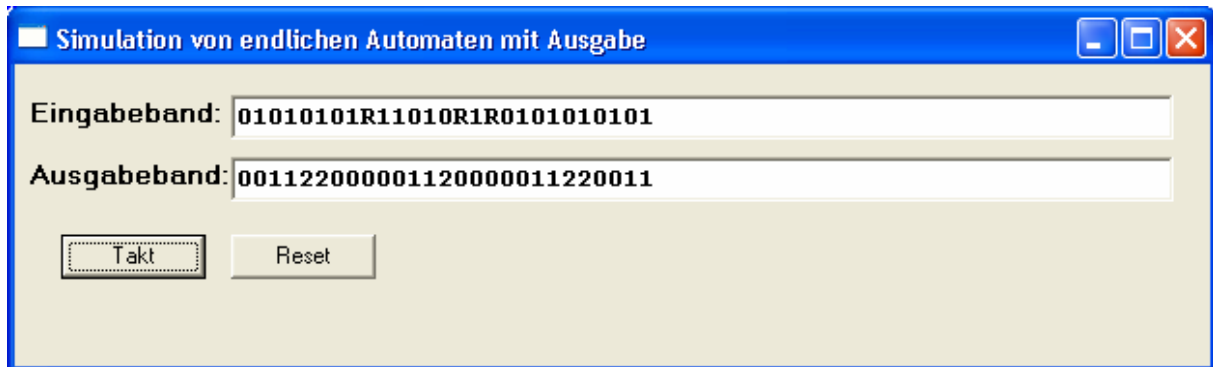


Zustandsmenge: $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$

3. Überprüfung durch ein Programm

Wir überprüfen die Funktionsweise des Automaten jetzt durch ein Java-Programm unter Benutzung der Windows Foundation Classes. Zustände bezeichnen wir mit ganzen Zahlen:

$$s_0 \rightarrow 0, s_1 \rightarrow 1, \dots$$



```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;

public class Form1 extends Form
{
    // Zustände werden mit ganzen Zahlen bezeichnet
    int i=-1,s=0;
    char rein,raus;

    //... hier steht von J++ erzeugter Code

    private void neu()
    {
        s = 0;
        i = -1;
        eEingabe.setText("");
        eAusgabe.setText("");
    }

    public int u(int s, char e)
    {
        switch(s)
        {
            case 0:
                switch(e) {case '0': return 0; case '1':return 1; case 'R': return 0;}
            case 1:
                switch(e) {case '0': return 2; case '1':return 1; case 'R': return 0;}
            case 2:
                switch(e) {case '0': return 2; case '1':return 3; case 'R': return 0;}
            case 3:
                switch(e) {case '0': return 4; case '1':return 3; case 'R': return 0;}
            case 4:
                switch(e) {case '0': return 4; case '1':return 5; case 'R': return 0;}
            case 5:
                switch(e) {case '0': return 1; case '1':return 5; case 'R': return 0;}
            default: return 99;
        }
    }
}
```

```
public char g(int s, char e)
{
    switch(s)
    {
        case 0: switch(e)
            {case '0': return '0'; case '1':return '0'; case 'R': return '0';}
        case 1: switch(e)
            {case '0': return '1'; case '1':return '0'; case 'R': return '0';}
        case 2: switch(e)
            {case '0': return '1'; case '1':return '1'; case 'R': return '0';}
        case 3: switch(e)
            {case '0': return '2'; case '1':return '1'; case 'R': return '0';}
        case 4: switch(e)
            {case '0': return '2'; case '1':return '2'; case 'R': return '0';}
        case 5: switch(e)
            {case '0': return '0'; case '1':return '2'; case 'R': return '0';}
        default: return ' ';
    }
}

private void bTakt_click(Object source, Event e)
{
    i++;
    if( i < eEingabe.getText().length() )
    {
        rein = eEingabe.getText().charAt(i);
        raus = g(s,rein);
        eAusgabe.setText(eAusgabe.getText()+raus);
        s = u(s,rein);
    }
}

private void bReset_click(Object source, Event e)
{
    neu();
}
```

//... hier steht von J++ erzeugter Code

4. Aufstellen der Schaltwerttabelle

Zuerst werden die benutzten Größen codiert. Die Codierung ist eigentlich beliebig, sollte aber so gewählt werden, dass man den Bits nachher ansieht, was sich dahinter verbirgt. Dadurch vermeidet man viele „Schreibfehler“.

Eingabealphabet:

Eingabezeichen	e ₀	e ₁
0	0	0
1	0	1
R	1	0

Ausgabealphabet:

Ausgabezeichen	a ₀	a ₁
0	0	0
1	0	1
2	1	0

Zustandsmenge:

Zustand	z ₀	z ₁	z ₂
s ₀	0	0	0
s ₁	0	0	1
s ₂	0	1	0
s ₃	0	1	1
s ₄	1	0	0
s ₅	1	0	1

In dieser Codierung wird der Transitionsgraph in die Schaltwerttabelle übertragen: Für jeden Zustand und jedes Eingabezeichen wird aufgeschrieben, welcher Folgezustand und welches Ausgabezeichen sich ergeben. Dabei werden nur die tatsächlich auftretenden Kombinationen berücksichtigt – das sind ja schon genug!

Zustand			Eingabezeichen		Folgezustand			Ausgabezeichen	
z ₀	z ₁	z ₂	e ₀	e ₁	u ₀	u ₁	u ₂	g ₀	g ₁
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0	1
0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0	1
0	1	0	0	1	0	1	1	0	1
0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	1	0	0	1	0
0	1	1	0	1	0	1	1	0	1
0	1	1	1	0	0	0	0	0	0
1	0	0	0	0	1	0	0	1	0
1	0	0	0	1	1	0	1	1	0
1	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	1	1	0
1	0	1	1	0	0	0	0	0	0


```

{
  e1 = false;
  do
  {
    //Schaltfunktionen berechnen
    u0 = ((z1 && z2) || (z0 && !z2)) && !e0 && !e1) ||
        (z0 && e1);
    u1 = ((!z1 && z2) || (z1 && !z2)) && !z0 && !e0 && !e1)
        || (z1 && e1);
    u2 = e1;
    g0 = u0;
    g1 = ((!z0 && (z1 ^z2)) && !e0) || (z1 && z2 && e1);

    // und in das Stringarray schreiben
    if(z0) h = " 1  |"; else h = " 0  |";
    if(z1) h = h + " 1  |"; else h = h + " 0  |";
    if(z2) h = h + " 1  |"; else h = h + " 0  |";
    if(e0) h = h + " 1  |"; else h = h + " 0  |";
    if(e1) h = h + " 1  ||"; else h = h + " 0  ||";
    if(u0) h = h + " 1  |"; else h = h + " 0  |";
    if(u1) h = h + " 1  |"; else h = h + " 0  |";
    if(u2) h = h + " 1  |"; else h = h + " 0  |";
    if(g0) h = h + " 1  |"; else h = h + " 0  |";
    if(g1) h = h + " 1"; else h = h + " 0";

    zeilen[i] = h;
    i++;
    e1 = !e1;
    // nur die vorkommenden Kombinationen anzeigen
    if (e0 && e1)
    {
      zeilen[i] =
        "-----";
      i++;
      break;
    }
  }
  while (e1 != false);
  e0 = !e0;
}
while(e0 != false);
z2 = !z2;
// nur die vorkommenden Kombinationen anzeigen
if (z0 && z1) break;
}
while(z2 != false);
z1 = !z1;
// nur die vorkommenden Kombinationen anzeigen
if (z0 && z1) break;
}
while(z1 != false);
z0 = ! z0;
// nur die vorkommenden Kombinationen anzeigen
if (z0 && z1) break;
}
while (z0 != false);

//Ergebnis anzeigen
richEdit1.setLines(zeilen);
}

```

// ... hier steht von J++ erzeugter Code

Das Ergebnis entspricht unseren Erwartungen – welch ein Glück!

Schaltwerttabelle										
z0	z1	z2	e0	e1	u0	u1	u2	g0	g1	

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0

0	0	1	0	0	0	1	0	0	0	1
0	0	1	0	1	0	0	1	0	0	1
0	0	1	1	0	0	0	0	0	0	0

0	1	0	0	0	0	1	0	0	0	1
0	1	0	0	1	0	1	1	0	0	1
0	1	0	1	0	0	0	0	0	0	0

0	1	1	0	0	1	0	0	1	0	0
0	1	1	0	1	0	1	1	0	0	1
0	1	1	1	0	0	0	0	0	0	0

1	0	0	0	0	1	0	0	1	0	0
1	0	0	0	1	1	0	1	1	0	0
1	0	0	1	0	0	0	0	0	0	0

1	0	1	0	0	0	0	0	0	0	0
1	0	1	0	1	1	0	1	1	0	0
1	0	1	1	0	0	0	0	0	0	0

7. Test im Digitalsimulator

Jetzt können wir die Gatterdarstellung der Schaltung im Digitalsimulator überprüfen. Es empfiehlt sich, die Frequenz des Taktgenerators soweit herunter zu setzen, dass die einzelnen Bausteine genügend Zeit haben, um zu schalten.

