Chemisches Automatenmodell eines Pseudozufallszahlengenerators

Blum-Blum-Shub

Molekulare Algorithmen

Friedrich-Schiller-Universität Jena

Gerd Specht & Timo Leistner

2021–08–09

Inhaltsverzeichnis

1 Aufgabenstellung

Ziel dieses Projekts ist die Implementierung des Blum Blum Shub-Pseudozufallszahlengenerators (auch $s^2 \mod n$-Generator) in einem chemischen Automatenmodell. Die Übersetzung dieses Algorithmus in ein chemisches Digitalcomputermodell weist verschiedene Herausforderungen auf.

So ist es notwendig eine möglichst effiziente Startkonfiguration zu ermitteln, die die Ausgabe einer Zahlensequenz mit Periodenlänge ≥ 21 bewirkt. s stellt den Zustand des Automaten dar, welcher mit jedem Takt i den Wert $s(i)$ mit $s(i+1)$ ersetzt. Nach vollständigem Durchlauf durch die Periode, soll sich die Ausgabe der Zufallszahlensequenz wiederholen. Dies erfordert die Implementierung eines Taktgebers als auch die Nachbildung der benötigten arithmetischen Operationen durch binäre logische Operationen.

In den folgenden Kapiteln werden wir sowohl die grundlegende Funktionsweise des Blum-Blum-Shub-Generators als auch dessen Implementierung in einem chemischen Automatenmodell erläutern und anhand einer Simulationsstudie in der Simulations-Software COPASI dessen Funktionsfähigkeit zeigen.

2 Blum-Blum-Shub

Der Blum-Blum-Shub-Generator ist ein simpler Algorithmus zur Erzeugung von Pseudozufallszahlen und gilt als kryptografisch sicher. Er wurde 1986 von Lenore Blum, Manuel Blum und Michael Shub entwickelt. Einsatz findet er bei der Erstellung kryptografischer Schlüssel.

Für die Initialisierung wird zuerst ein *random seed* als Startwert s_0 benötigt. Desweiteren werden zwei Primzahlen p und q benötigt welche folgende Bedingung erfüllen müssen:

$$p, q \equiv 3 \pmod 4.$$

Aus diesen lässt sich n (auch *Blum-Zahl* genannt) berechnen:

$$n = p * q.$$

Die Sequenz wird dann wie folgt generiert:

$$s_{i+1} \equiv s_i^2 \pmod n.$$

Wobei der Startwert s_0 zu n teilerfremd sein muss. [1]

Die hier realisierte Implementierung des Blum-Blum-Shub-Generators benutzt die Werte $s_0 = 2, p = 19$ und $q = 23$. In dieser Konfiguration wird eine Zufallszahlensequenz mit einer Periode von 30 generiert. Die Minimalwerte für s_0, p und n wurden mit den Skripten `bitstrings_table.jl` und `blumblumshub.jl` ermittelt.

Eine simple Implementierung in der Programmiersprache Julia:

```
function bbshub(s, p, q)
    sequence = [s]
    flag = true
    while flag == true
        s = s^2 % (p * q)
        if s in sequence
            flag = false
        end
        push!(sequence, s)
    end
    return sequence
end
```

3 Vorgehensweise

3. a) 5-Bit Gray-Code

Um eine möglichst robuste Signalübertragung zu gewährleisten und um die Implementierung so simpel wie möglich zu halten wird ein nach Frank Gray benannter Gray-Code verwendet. Aufeinander folgende Bitfolgen unterscheiden sich somit möglichst immer nur um 1 Bit (Hamming-Distanz von 1).

3. b) Boolesche Funktionen

Die Booleschen Funktionen für die entsprechenden Bits können aus Tabelle 1 oder 2 abgelesen werden. Um die Funktionen und somit die darauf aufbauende Implementierung so simpel wie möglich zu halten wurden diese per Karnaugh-Optimierung minimiert.

3. c) Karnaugh-Optimierung

Um die Booleschen Funktionen aus Tabelle 2 zu vereinfachen wurde ein SOP (Sum of Products) Algorithmus verwendet [3].
Dabei ergaben sich folgende Vereinfachungen:

$$
\begin{aligned}
b'_1 = 1 &\iff (b_1 b_4) \vee (b_1 b_3) \vee (b_1 b_2) \vee (b_2 \overline{b_3 b_4 b_5}) \\
b'_2 = 1 &\iff (b_2 \overline{b_3}) \vee (b_2 b_5) \vee (b_2 b_4) \vee (\overline{b_1} b_3 \overline{b_4 b_5}) \\
b'_3 = 1 &\iff (b_3 \overline{b_4}) \vee (b_3 b_5) \vee (\overline{b_1 b_2} b_4 \overline{b_5}) \vee (b_1 b_2 b_4 \overline{b_5}) \\
b'_4 = 1 &\iff (b_4 \overline{b_5}) \vee (\overline{b_1 b_2 b_3} b_5) \vee (\overline{b_1} b_2 b_3 b_5) \vee (b_1 \overline{b_2} b_3 b_5) \vee (b_1 b_2 \overline{b_3} b_5) \\
b'_5 = 1 &\iff (\overline{b_1 b_2 b_3 b_4}) \vee (\overline{b_2 b_3 b_4} b_5) \vee (\overline{b_1 b_2} b_3 b_4) \vee (\overline{b_1} b_2 \overline{b_3} b_4) \\
&\quad \vee (\overline{b_1} b_2 b_3 \overline{b_4}) \vee (b_1 \overline{b_2 b_3} b_4) \vee (b_1 \overline{b_2} b_3 \overline{b_4}) \vee (b_1 b_2 \overline{b_3 b_4}) \vee (b_1 b_2 b_3 b_4)
\end{aligned}
$$

$$
\begin{aligned}
z_1 = 1 &\iff (b_2 b_3 \overline{b_4}) \vee (b_1 b_2 b_3) \vee (\overline{b_1 b_2 b_4} b_5) \vee (\overline{b_1 b_2} b_3 b_4) \vee (b_1 \overline{b_3} b_4 \overline{b_5}) \\
&\quad \vee (b_1 b_3 \overline{b_4 b_5}) \vee (\overline{b_1} b_2 \overline{b_3} b_4 b_5) \\
z_2 = 1 &\iff (\overline{b_1} b_2 b_4) \vee (\overline{b_1} b_2 b_3) \vee (b_2 b_3 b_5) \vee (b_1 \overline{b_2} b_3) \vee (\overline{b_1} b_3 b_4 \overline{b_5}) \\
&\quad \vee (b_1 \overline{b_2} b_4 b_5) \vee (b_1 b_2 \overline{b_4} b_5) \vee (\overline{b_1 b_2 b_3 b_4 b_5}) \\
z_3 = 1 &\iff (\overline{b_2} b_3 \overline{b_4}) \vee (\overline{b_2} b_3 \overline{b_5}) \vee (b_1 b_3 b_4 \overline{b_5}) \vee (\overline{b_1} b_2 \overline{b_3 b_4} b_5) \vee (\overline{b_1} b_2 \overline{b_3} b_4 \overline{b_5}) \\
&\quad \vee (\overline{b_1} b_2 b_3 b_4 b_5) \vee (b_1 b_2 \overline{b_3} b_4 b_5)
\end{aligned}
$$

Tabelle 1: Schaltbelegungstabelle

Zählwert	b_1	b_2	b_3	b_4	b_5	b'_1	b'_2	b'_3	b'_4	b'_5	z_1	z_2	z_3
2	0	0	0	0	0	0	0	0	0	1	0	1	0
4	0	0	0	0	1	0	0	0	1	1	1	0	0
16	0	0	0	1	1	0	0	0	1	0	0	0	0
256	0	0	0	1	0	0	0	1	1	0	0	0	0
423	0	0	1	1	0	0	0	1	1	1	1	1	1
196	0	0	1	1	1	0	0	1	0	1	1	0	0
397	0	0	1	0	1	0	0	1	0	0	1	0	1
289	0	0	1	0	0	0	1	1	0	0	0	0	1
54	0	1	1	0	0	0	1	1	0	1	1	1	0
294	0	1	1	0	1	0	1	1	1	1	1	1	0
347	0	1	1	1	1	0	1	1	1	0	0	1	1
234	0	1	1	1	0	0	1	0	1	0	0	1	0
131	0	1	0	1	0	0	1	0	1	1	0	1	1
118	0	1	0	1	1	0	1	0	0	1	1	1	0
377	0	1	0	0	1	0	1	0	0	0	0	0	1
104	0	1	0	0	0	1	1	0	0	0	0	0	0
328	1	1	0	0	0	1	1	0	0	1	0	0	0
82	1	1	0	0	1	1	1	0	1	1	0	1	0
169	1	1	0	1	1	1	1	0	1	0	0	0	1
156	1	1	0	1	0	1	1	1	1	0	1	0	0
301	1	1	1	1	0	1	1	1	1	1	1	0	1
142	1	1	1	1	1	1	1	1	0	1	1	1	0
62	1	1	1	0	1	1	1	1	0	0	1	1	0
348	1	1	1	0	0	1	0	1	0	0	1	0	0
55	1	0	1	0	0	1	0	1	0	1	1	1	1
403	1	0	1	0	1	1	0	1	1	1	0	1	1
282	1	0	1	1	1	1	0	1	1	0	0	1	0
427	1	0	1	1	0	1	0	0	1	0	0	1	1
100	1	0	0	1	0	1	0	0	1	1	1	0	0
386	1	0	0	1	1	1	0	0	0	1	0	1	0
416	1	0	0	0	1	0	0	0	0	1	0	0	0

Tabelle 2: Wahrheitswerttabelle für die Eingabe der Karnaugh-Optimierung

#	A	B	C	D	E	b'_1	b'_2	b'_3	b'_4	b'_5	z_1	z_2	z_3
0	0	0	0	0	0	0	0	0	0	1	0	1	0
1	0	0	0	0	1	0	0	0	1	1	1	0	0
2	0	0	0	1	0	0	0	1	1	0	0	0	0
3	0	0	0	1	1	0	0	0	1	0	0	0	0
4	0	0	1	0	0	0	1	1	0	0	0	0	1
5	0	0	1	0	1	0	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	1	1	1	1
7	0	0	1	1	1	0	0	1	0	1	1	0	0
8	0	1	0	0	0	1	1	0	0	0	0	0	0
9	0	1	0	0	1	0	1	0	0	0	0	0	1
10	0	1	0	1	0	0	1	0	1	1	0	1	1
11	0	1	0	1	1	0	1	0	0	1	1	1	0
12	0	1	1	0	0	0	1	1	0	1	1	1	0
13	0	1	1	0	1	0	1	1	1	1	1	1	0
14	0	1	1	1	0	0	1	0	1	0	0	1	0
15	0	1	1	1	1	0	1	1	1	0	0	1	1
16	1	0	0	0	0	0	0	0	0	0	0	0	0
17	1	0	0	0	1	0	0	0	0	1	0	0	0
18	1	0	0	1	0	1	0	0	1	1	1	0	0
19	1	0	0	1	1	1	0	0	0	1	0	1	0
20	1	0	1	0	0	1	0	1	0	1	1	1	1
21	1	0	1	0	1	1	0	1	1	1	0	1	1
22	1	0	1	1	0	1	0	0	1	0	0	1	1
23	1	0	1	1	1	1	0	1	1	0	0	1	0
24	1	1	0	0	0	1	1	0	0	1	0	0	0
25	1	1	0	0	1	1	1	0	1	1	0	1	0
26	1	1	0	1	0	1	1	1	1	0	1	0	0
27	1	1	0	1	1	1	1	0	1	0	0	0	1
28	1	1	1	0	0	1	0	1	0	0	1	0	0
29	1	1	1	0	1	1	1	1	0	0	1	1	0
30	1	1	1	1	0	1	1	1	1	1	1	0	1
31	1	1	1	1	1	1	1	1	0	1	1	1	0

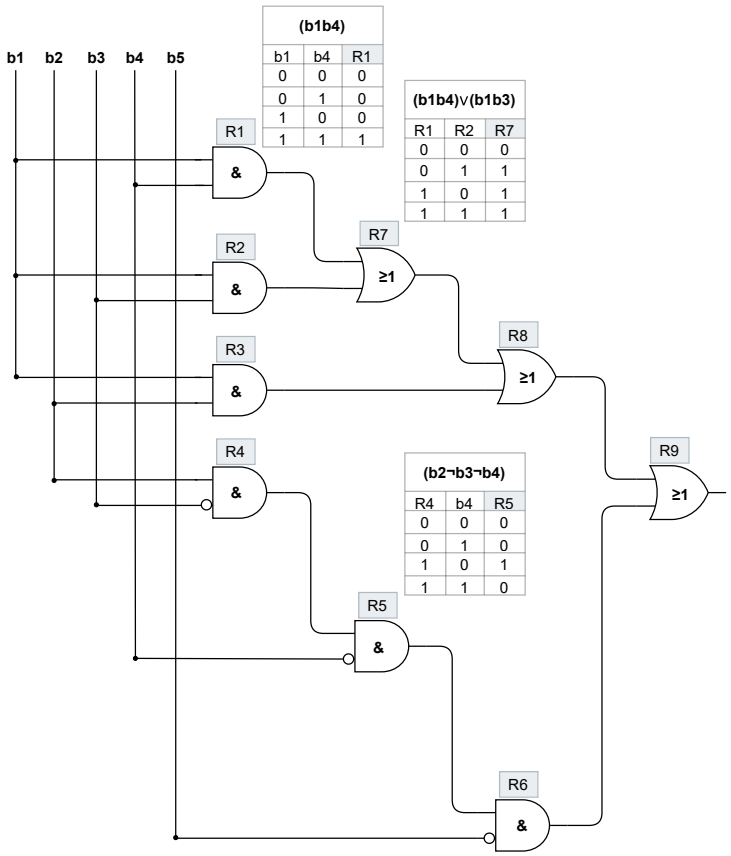

Abbildung 1: Netzwerkmodell der Schaltung von b_1'

3. d) Brüsselator als Taktgeber

Der Brüsselator ist ein chemisches Modell für eine autokatalytische positiv rückgekoppelte Reaktion, die unter bestimmten Bedingungen oszilliert. Der Grundaufbau des Brüsselators lässt sich in chemischen Reaktionen wie folgt beschreiben:

$$
\begin{aligned}
A &\to X \\
2X + Y &\to 3X \\
B + X &\to Y + D \\
X &\to E
\end{aligned}
$$

Der Brüsselator erzeugt eine spikeförmige Oszillation. Für eine Nutzung als Taktgeber in einem chemischen Digitalcomputer, sollte die Oszillation möglichst stufenförmig ablaufen. Daher wird der von uns verwendete Oszillator mithilfe von Wellentranformationen in eine Stufenform überführt.

3. e) COPASI Implementierung

Das chemische Automatenmodell besteht aus dem Brusselator-Taktgeber, den fünf optimierten Booleschen Funktionen $b'_1, b'_2, b'_3, b'_4, b'_5$ und den drei Ausgabebits z_1, z_2, z_3. Die Pseudozufallszahlen-Sequenz beinhaltet Werte bis maximal 472. Um Zahlen bis einschließlich 427 zu repräsentieren wären 9 Bits nötig (kodiert bis maximal 512). Um die Implementierung zu simplifizieren verwenden wir lediglich die niederen 3 Bits dieser generierten Zahlen (z_1, z_2, z_3). Die korrektheit kann dann durch die Folge der jeweils 3 Bits überprüft werden.

Insgesamt beläuft sich die Zahl der zur Implementierung notwendigen Reaktionen auf 1148, welche mit 389 chemischen Spezies gebildet wurden. Der Brusselator konnte aus einem vorausgegangenen Studentenprojekt übernommen werden. Die Spezies `03T` wird in diesem Modell als Taktgeber für alle Reaktionen verwendet und wird dabei nicht verbraucht, sondern als Katalysator behandelt. Die optimierten Booleschen Funktionen wurden als kaskadierende Und/Oder-Gatter realisiert (siehe Abbildung 1). Die Signalkaskaden verlangsamen zwar das Reaktionsnetzwerk, jedoch kann so die Zahl der benötigten Reaktionen auch möglichst gering gehalten werden. Hierbei wird jedes Und-Gatter mithilfe von 4 Reaktionen, welche alle möglichen Eingabe-Kombinationen abdecken und 2 sogenannten Verstärkungsreaktionen, die die Signalstärke (d.h. Konzentration der beteiligten Spezies) erhöhen. Die Oder-Gatter bestehen ebenfalls aus 4 Reaktionen, die die Logik beschreiben und 2 weiteren Verstärkungsreaktionen.

Beispielsweise ist das in Abbildung 1 zu sehende Und-Gatter $R1$ wie folgt in Copasi in Form von Reaktionsgleichungen implementiert:

```
R1T + b1F + b4F + O3T -> R1F + b1F + b4F + O3T
R1T + b1F + b4T + O3T -> R1F + b1F + b4T + O3T
R1T + b1T + b4F + O3T -> R1F + b1T + b4F + O3T
R1F + b1T + b4T + O3T -> R1T + b1T + b4T + O3T
R1T + 2 * R1F -> 3 * R1F
R1F + 2 * R1T -> 3 * R1T
```

Die Flipflop-Komponenten wurden aus dem vorausgegangenen Frequency Divider-Projekt übernommen und ermöglichen das Selbstfeedback für die Booleschen Funktionen.

Die 3 Ausgabebits werden ähnlich wie die 5 optimierten Booleschen Funktionen realisiert, jedoch fehlt hier das Selbstfeedback.

4 COPASI Ergebnisse

Die Konzentrationen der Spezies `R9`, `R18`, `R29`, `R46` und `R81` beschreiben jeweils die Zustände der Bits $b'_1, b'_2, b'_3, b'_4, b'_5$. Deren Konzentrationsverhalten kann in den Abbildungen 2 und 3 beobachtet werden.

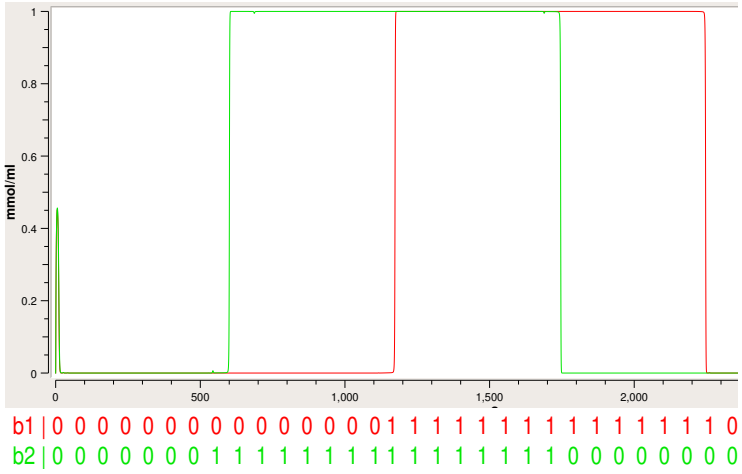

Abbildung 2: Konzentrationsverlauf der Spezies $R9$ und $R18$ für die Bits b'_1, b'_2.

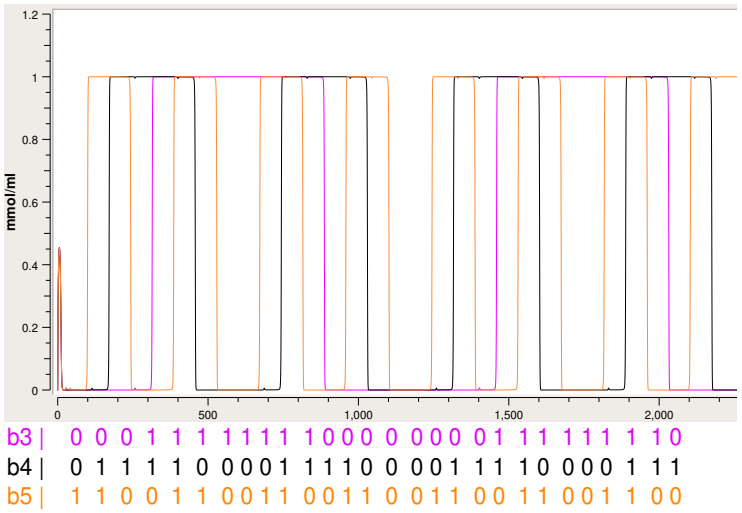

Abbildung 3: Konzentrationsverlauf der Spezies $R29, R46$ und $R81$ der Bits b'_3, b'_4, b'_5.

Die Konzentrationen der Spezies `R107T`, `R135T` und `R164T`, beschreiben die Zustände der Bits z_1, z_2 und z_3.

Nach Abschluss der Sequenz beginnt diese beim Zählwert 4 erneut (Zählwert 2 taucht lediglich beim ersten Durchlauf auf, da diese den *seed* darstellt).

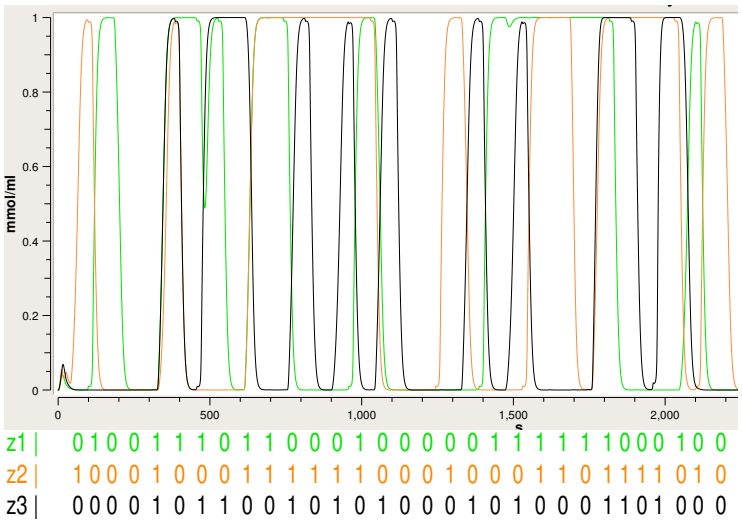

Abbildung 4: Konzentrationsverlauf der niederen 3 Bits der generierten Pseudo-zufallszahlen.
z_1, z_2, z_3.

Quellen

[1] L. Blum, M. Blum und M. Shub. “A Simple Unpredictable Pseudo-Random Number Generator”. In: *SIAM Journal on Computing* 15.2 (Mai 1986), S. 364–383. ISSN: 0097-5397. DOI: 10.1137/0215025.

[2] Thomas Hinze. *Computer Der Natur*.

[3] *Online Karnaugh Map Solver with Circuit for up to 6 Variables*. http://www.32x8.com/index.html.