Pseudozufallszahlen

Assignment zum Modul:

Digitaltechnik (ELT03)

13. September 2018

Vladimir Zhelezarov

.....

Studiengang: Digital Engineering und angewandte Informatik - Bachelor of Engineering

AKAD University

Inhaltsverzeichnis

1 Einleitung

In verschieden Situationen ist er notwendig, schnell und effektiv eine Sequenz aus zufällige Zahlen zu erstellen. Die bekannteste Beispiele sind die Kryptographie, die Kommunikation - wie im Internet und bei der GSM Technologie sowie bei überraschend viele andere Anwendungen. Zum Beispiel ist die fast überall benutzte Fehlerkontrolle der Datenübertragung bekannt als CRC (cyclic redundancy check) im Grunde nach dem selben mathematischen Rahmen aufgebaut und hat fast die gleiche Schaltung, wie der hier beschriebene Zufallszahlgenerator. So ist die Thema dieser Arbeit ein wichtiger Grund für verschieden Einsätze.

Wir werden uns mit „normale“, also nicht für kryptographische Zwecke Zufallszahlen beschäftigen, die auch deterministisch, sprich vorhersehbar, sind (vorausgesetzt man kennt einige Parameter). Es wird ein basierend auf Schieberegister Pseudozufallszahlengenerator vorgestellt, als eine schnelle und einfach-zu-bauende Lösung, die auch sehr verbreitet und benutzt ist. Der mathematische Rahmen, der dafür benötigt wird, basiert auf der Arbeit von Évariste Galois (französischer Mathematiker 1811-1832).

Dafür definieren wir zuerst, was wir unter Zufallszahlen und Zufallszahlengeneratoren verstehen und was dabei zu beachten ist. Danach sehen wir was für Möglichkeiten es gibt, damit geeignete Sequenzen zu erstellen. Kurz stellen wir auch vor, was für Bauelemente dafür notwendig werden.

Um die finale Schaltung der Arbeit (mit 16 Bit und eine Periodenlänge von 65 535) besser zu verstehen, fangen wir mit einem einfachsten 4-Bit Schieberegister an und dann bauen wir auf, wobei wir die Schaltung mit weitere Bits, Einstellungsmöglichkeiten und Fehleranzeige schrittweise erweitern. Um die Größe der dafür nötigen Abbildungen in Grenzen zu halten, wird hier ein 16 Bit Generator dargestellt. Allerdings können nach Bedarf und Verfügbarkeit durch das vorgestellte Schema beliebig große Schaltungen gebaut werden.

Als eine hervorragend geeignete Anwendung wird danach ein Bitfehlertester betrachtet, sowie auch andere potenzielle Einsätze der Schaltung.

2 Grundlagen

2.1 Zufallszahlen

Allgemein definiert ist eine Reihenfolge aus Nummer zufällig, wenn ein Beobachter schwer oder gar nicht die nächste Zahl in der Folge raten kann. Die Statistik beschäftigt sich mit den Besonderheiten der Verteilung von solche Zahlen. Vereinfacht kann man sagen, dass die Verteilung von alle mögliche Zahlen in einer Zufallssequenz gleich ist[1].

Man unterscheidet „echte“ und „pseudo-“ Zufallszahlengeneratoren[2]:

- Bei den ersten (aus Englisch: TRNG True Random Number Generator) kann man nicht wissen, oder berechnen was als nächstes kommt - selbst wenn man den Ausgangszustand und die Funktionsweise des Generators kennt. Als Beispiel lassen sich hier Umwelteinflüsse wie Umgebungsstörungen, Lärm, elektrische Schwankungen, analoge Signale o.ä. geben. Es versteht sich von alleine, dass echter Zufall besonders schwierig zu generieren und vor allem zu beweisen ist. Oftmals ist das auch nicht benötigt. So kommen wir auf den zweiten Fall von Generatoren:

- Pseudozufallszahlengeneratoren (aus Englisch: PRNG Pseudo Random Number Generator). Von diesen wird eine Folge von Zahlen erstellt, deren Verteilung statistisch gesehen sehr nah oder nicht unterschiedlich ist von einer Zufallsverteilung. Im Vergleich zu den TRNG, sind aber die Zahlen bei einem PRNG völlig vorhersehbar und berechenbar, solange man den Aufbau vom Generator und seinem Ausgangsstand kennt.

In dieser Arbeit fokussieren wir uns auf PRNG.

2.2 Pseudozufallszahlengenerator

Bei Definition, um die nächst-kommende Zahl bei einem PRNG zu berechnen, braucht man zwei Informationen: Ausgangszustand vom System und den Algorithmus / die Formel, wonach die Zahlen kalkuliert werden.

Der Wert, womit man den PRNG initialisiert, wird „Seed“ (aus Englisch für „Saat“) genannt. Dadurch kommt es, das der gleiche PRNG verschiedene Zahlenfolgen erstellt, wenn er zum Start mit verschiedene Seeds gefüttert wurde.

Andere wichtige Eigenschaft eines PRNG ist seine Periode - das ist schließlich die Zahl von verschiedene Nummer, die er generieren kann, bevor die Sequenz an fängt sich zu wiederholen. Also erwünscht sind PRNG mit möglichst große Perioden. Das erzielt man durch gute Auswahl vom Algorithmus bezogen auf die vorhandene Hardware.

Es kann auch vorkommen, dass für bestimmte Kombination von Hardware und Algorithmus, einige Zahlen für Seed als verboten zählen, weil sie zu einem unerwünschten Zustand

[1] Vgl. Viega, J. (o.J.), Internetquelle
[2] Vgl. Viega, J. (o.J.), Internetquelle

oder Fehlfunktion vom Generator führen, wie wir gleich bei unseren Schieberegister Generator sehen werden.

2.3 Benötigte Bauelemente

Um unser PRNG zu realisieren, werden wir folgende Bausteine benutzen:

- Logische Verknüpfungen wie AND, OR, NOT etc.
- Ein- und Ausgänge zum Eingeben oder Ablesen von den Daten. Wegen Übersichtlichkeit werden in der Simulationssoftware als Ausgang einfach Lampen benutzt;
- Darauf aufbauend folgen die Flip-Flops: logische Elemente, die ihr Zustand speichern können[1]. Von den verschiedene existierende Versionen wird hier das D Flip-Flop benutzt - das ist ein sehr einfach zu bedienendes Flip-Flop, das nur ein Eingang zum Set/Reset benutzt, wessen Wert, während der Clock sein hohen Pegel hat, abgelesen wird. Sein Nachteil liegt daran, dass der Eingang sich nicht verändern darf, während der Clock „hoch“ ist[2]. Bei der vorgeschlagene Bauweise ist dieses Problem so gut wie ausgeschlossen, darum wird das D Flip-Flop wegen seiner einfacheren Benutzung und der Steuerung über eine Leitung genommen.

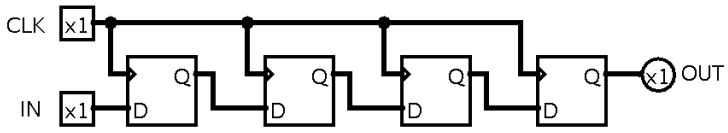
2.4 Simulation

In dieser Arbeit wird Logisim als Simulationssoftware benutzt. Logisim ist verfügbar auf (http://www.cburch.com/logisim/de/index.html) unter GNU General Public License, Version 2.

3 Linear rückgekoppeltes Schieberegister

Schieberegister sind Schaltwerke aus logische Bausteine, meistens mit Flip-Flops realisiert[3], deren Glieder ihr Wert bei jedem Takt eine Position nach links oder nach rechts schieben[4], wie unsere Beispielabbildung zeigt:

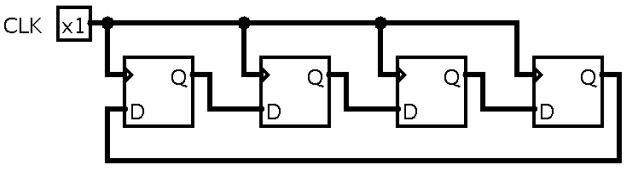
[1] Vgl. Koch et al. (o.J.), S.11
[2] Vgl. Koch et al. (o.J.), S.15
[3] Vgl. Saluja, K. (1991), S.1
[4] In dieser Arbeit beschäftigen wir uns nur mit Rechtsschieber, die Linksschieber folgen die gleiche Logik nur mit umgekehrter Richtung



Takt	Wert			
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

Abbildung 1: Schieberegister

Verbindet man den Ausgang mit dem Eingang, so kommt es dazu, dass die Werte durch das Schieberegister „kreiseln“ - also zum Beispiel beim Shiften nach rechts wird das letzte Bit (LSB - least significant bit) beim nächsten Takt zum ersten Bit: MSB (most significant bit). Bei jedem Takt kann man das Schieberegister ablesen und dadurch eine Zahl bekommen - wir sind auf dem Weg zum PRNG. So ein Schieberegister nennt man rückgekoppelt und wird durch die englische Abkürzung LFSR (linear feedback shift register) bezeichnet:



Takt	Wert			
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5(wie 1)	1	0	0	0

Abbildung 2: Linear rückgekoppeltes Schieberegister

Die oben gezeigte Abbildung ist eigentlich ein Sonderfall mit $C_1 = C_2 = C_3 = 0$ der allgemeine Darstellung von einem 4-Bit LFSR, die wie folgt aussieht:

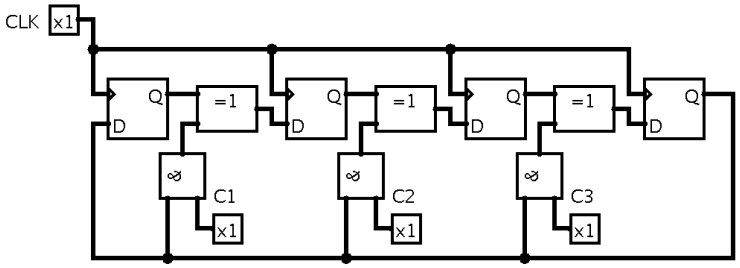


Abbildung 3: Linear rückgekoppeltes Schieberegister - allgemeine Darstellung

Es lässt sich mathematisch beweisen[1], dass diese Schaltung durch das Polynom $p(x) = x^4 + C_1.x^3 + C_2.x^2 + C_3.x + 1$ bezeichnet wird. Die binäre Koeffizienten C_n zeigen praktisch an, ob vor dem entsprechenden Gatter eine XOR Verknüpfung steht.

[1] Vgl. Saluja, K. (1991), S.9

Für bestimmte Polynomen wird die maximale Periodenlänge vom LFSR erreicht, die $P = 2^n - 1$ ist[1]. Um eine potenziell aufwändige Berechnung von solche Kombinationen zu vermeiden, werden oftmals vorgefertigte Tabellen benutzt[2]:

Bits	Polynom	Periode $(2^n - 1)$
2	$x^2 + x + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
...	...	...
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$	65535

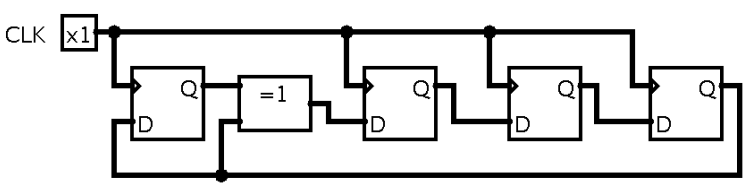
Tabelle 1: Manche optimale Polynome

So, um jetzt eine 4-Bit Schaltung zu berechnen, die auch die maximal mögliche Periodenlänge besitzt, entnehmen wir der Polynomentabelle, dass eine optimale Variante bei 4 Bits das Polynom $p(x) = x^4+x^3+1$ ist. Anders geschrieben ist das $p(x) = x^4+1.x^3+0.x^2+0.x+1$, wovon wir erkennen, dass $C_1 = 1$, $C_2 = 0$ und $C_3 = 0$. Also wir brauchen ein XOR Gatter vor dem Bit 3.

Der nächste Zustand (Zahl) können wir schon aus dem jetzigen Zustand und dem Polynom berechnen. Das Polynom stellen wir als eine binäre Maske dar, wenn wir eine „1“ einsetzen, da wo „x“ steht: $Maske = b1100$. Dann können wir die nächste Zahl so berechnen:

- Wenn das letzte Bit 1 ist, dann alle Bits ein Bit nach rechts schieben und darauf folgend mit der Maske XOR-en;
- Wenn nicht, dann nur alle Bits ein Bit nach rechts schieben.

Die Schaltung sieht schon so aus:



Takt	Wert			
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	1	0	0
6	0	1	1	0
..	..	..	..	..
14	1	0	1	1
15	1	0	0	1
16(wie 1)	1	0	0	0

Abbildung 4: Linear rückgekoppeltes Schieberegister mit XOR

[1] Vgl. Saluja, K. (1991), S.15
[2] Ward, R.; Molteno, T. (2007), Internetquelle

Die gezeigte Verbindung mit XOR Gatter zwischen den Flip-Flops entspricht den LFSR Aufbau nach Galois[1]. Alternativ kann man die Schaltung auch anders gestalten (wie beim sogenannten Fibonacci-LFSR)[2], das Ziel dieser Arbeit ist aber ein PRNG mittels Galois-Aufbau zu erzielen.

Für die beigefügte Tabellen mir Werten, haben wir bis jetzt angenommen, dass der Ausgangszustand von den Flip-Flops [1,0,0,0] ist. Allerdings haben wir immer noch keine Möglichkeit das einzustellen.

Zusätzlich existiert bei der gezeigte Konfiguration ein verbotener Zustand und zwar, wenn alle Bits Null sind. Dann spielt es keine Rolle, wie viele Takts vergangen sind - es verändert sich nichts, weil die Nullen beim Schieben und XOR-en auch Nullen bleiben. Dafür sind Maßnahmen zu treffen, um dies zu vermeiden. Weil diese Bitfolge nicht von alleine auftreten kann, soll es nur vermieden werden, dass sie vom Benutzer als Seed gegeben wird.

Über eine einfache Version von Multiplexer [3] kann entschieden werden, ob die Flip-Flops sich im „Normalbetrieb“ befinden, oder es gerade Seed eingestellt wird. So entsteht schon die Möglichkeit Seed einzugeben:

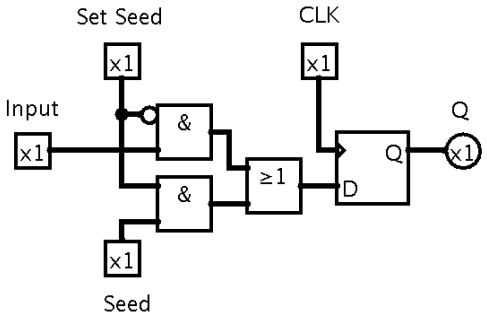


Abbildung 5: D Flip-Flop Erweiterung

Der Vereinfachung halber, definieren wir, dass die oben gezeigte Schaltung integriert wird und ein eigenes Zeichen bekommt, wie folgt:

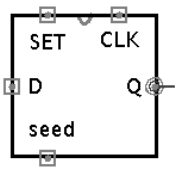


Abbildung 6: Schaltzeichen vom erweiterten D Flip-Flop

[1] Vgl. o.V. (2005), Internetquelle
[2] Vgl. o.V. (2005), Internetquelle
[3] Vgl. Koch (o.J.), S.46f.

Es fehlt noch nur eine Rückmeldung, wenn die Bits im verbotenen Zustand sind. Das kann über eine AND Verknüpfung mit negierte Eingänge erreicht werden. Damit kann die finale Schaltung von unserem PRNG mit Periodenlänge 15 so aussehen:

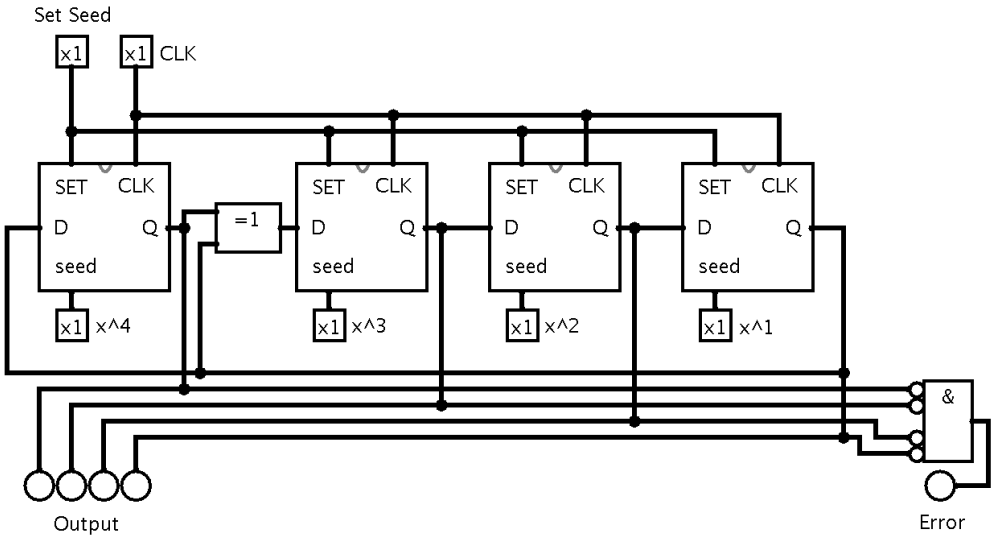


Abbildung 7: 4-Bit Schaltung mit Periodenlänge 15

4 Längere Perioden

Aufbauend auf der Schaltung mit vier Bits und die Tabelle mit den optimale Polynomen, kann man die Konfiguration auf Wunsch erweitern. Angenommen man bräuchte eine Periodenlänge von 65 535. Der Tabelle ist zu entnehmen, dass dies eine Registerbreite von 16 Bits entspricht und eine optimale Funktion (Polynom) dafür $p(x) = x^{16} + x^{14} + x^{13} + x^{11} + 1$ wäre. Also wir brauchen XOR-Gatter vor dem 14-ten, 13-ten und dem 11-ten Glied . Dann würde die Schaltung mit den allen von uns eingeführten Erweiterungen so aussehen:

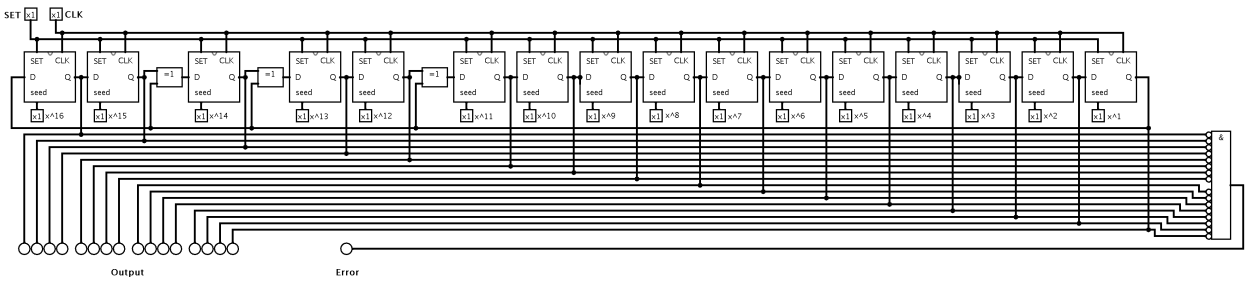


Abbildung 8: 16-Bit Schaltung mit Periodenlänge 65 535

5 Anwendungen

5.1 Bitfehlertester

Bei der Nachrichtenübertragung ist es oftmals notwendig eine neu-gebaute Schaltung oder ein Verbindungskanal gründlich zu testen, indem man viele verschiedene Bits dadurch sendet oder damit bearbeitet. Das Verhalten der Schaltung oder des Kanals sollte dabei natürlich der Spezifikation entsprechen. Eine Möglichkeit wäre das über einem verbundenem Computer mit der entsprechende Software durchzuführen, allerdings kommt dieses Verfahren schnell an seine Grenzen, was Geschwindigkeit und Speicherbedarf an geht, wenn die Schaltung größer wird. Die Einfache Bauweise eines LFSR führt dazu, dass er eine unschlagbare Effektivität bei der Erstellung von Bits erreichen kann.

Als einfache eingebaute Schaltung oder als externer Tester, ist ein LFSR als Bitfehlertester sehr gut geeignet. Von der eine Seite der Verbindung, oder des Eingangs der Verarbeitungsschaltung werden Bits generiert und von der andere, die angekommene Daten mit der Soll-Sequenz verglichen. Besteht irgendwo Unterschied, dann liegt ein Problem bei der Übertragung, bzw. bei der Überarbeitung der Daten vor.

Für die Empfängerseite, lässt sich die Schaltung so ergenzen:

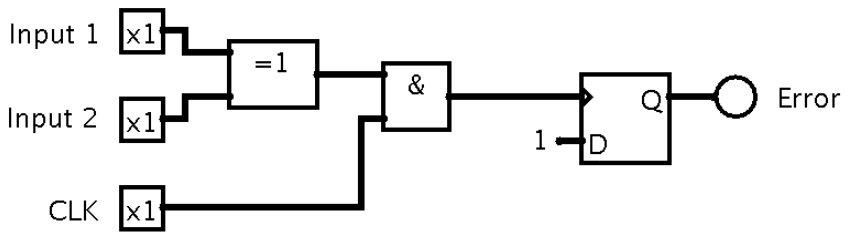


Abbildung 9: Komparator mir Fehlermeldung

Der eine Eingang kommt von einem gleichen LFSR, wie beim Sender. Der andere lest die angekommene Bits ab. Der XOR Gatter vergleicht seine Eingänge und gibt nur dann eine 1 aus, wenn sie sich unterscheiden. Der „CLK“ Eingang bestimmt die Frequenz dieser Operation. Danach ist das Flip-Flop, das auf sein Eingang permanent eine 1 hat. Dadurch reicht es, wenn es irgendwann die Eingänge sich unterscheiden, damit der Ausgang „Error“ permanent geschaltet wird.

Ohne zu tief in die Problematik zu gehen, lassen sich die bei dieser Schaltung potenzielle Probleme bei der Synchronisierung durch geeignete elektronische Regelungen des Clocks umgehen.

5.2 Andere Anwendungen

Ohne Anspruch auf Vollständigkeit sind hier manche von den bekanntesten andere Einsatzgebiete vom LFSR gelistet:

- Als Grundschaltung für die CRC Fehlerkontrolle. Im Internet werden praktisch alle Pakete mit CRC-Summe versehen, um Übertragungsprobleme zu erkennen;
- Als Scrambler/Descrambler. Bei der Funksendung von Daten ist es lieber, dass die Datenmenge nicht auf Wellen gesendet wird, sondern dass die Antenne und die dazugehörige Schaltungen gleichmäßig belastet werden. Dafür mischt man das Signal mit einer Sequenz aus LFSR. Von der Empfängerseite lässt sich einfach und schnell mit demgleichen LFSR diese Operation umkehren;
- Stromchiffre: Bei der Kryptographie kombiniert man manchmal mehrere LFSR mit sorgfältig gewählte Polynome. Die Daten werden laufend verschlüsselt, womit sie mit der erzeugte Sequenz gemischt werden. Entschlüsseln lässt sich der Stream einfach mit umgekehrte Anwendung von der gleiche Schaltung;
- Als Zähler. Manchmal spielt es keine Rolle, wenn es z.B. bis 100 gezählt werden muss, ob es 1,2,3.. gezählt wird oder 84,2, 6.. solange die Gesamtmenge von Zahlen die selbe ist. Bei diesem Einsatz wird wieder die enorme Geschwindigkeit vom LFSR ausgenutzt.

6 Zusammenfassung

Ob als Scrambler, oder Schaltungstester, oder als Teil einer CRC - auf ein Pseudozufallszahlengenerator kann in der moderne Welt nicht verzichtet werden. Die von ihm generierte Sequenzen von Zahlen haben die interessante Eigenschaften, dass sie gleichzeitig wie vom Zufall erstellt aussehen, können aber trotzdem schnell und problemlos im Voraus berechnet werden. Dafür braucht man nur den Aufbau, oder anders gesagt das Polynom des Generators kennen sowie auch den letzten Zustand.

Basierend auf einfache und verfügbare Bausteine, wie logische Gatter und D Flip-Flops, wurde hier ein Pseudozufallszahlengenerator nach dem mathematischen Rahmen von Galois vorgestellt. Um beliebig lange Perioden zu erreichen, fängt man mit einem einfachen Schieberegister an, indem man sein Ausgang zu seinem Eingang hinführt. Durch zufügen von XOR Gatter an gut ausgewählte Plätze zwischen der Schaltwerke kann man die volle mögliche Periodenlänge für die verfügbare Hardware erreichen. Sie beträgt $(2^n - 1)$. Weil die Berechnung aufwendig ist, benutzt man vorgefertigte Tabellen mit optimale Polynomen. Zu beachten bei der vorgestellte Schaltung ist der „verbotene Zustand“ - wenn der Seed aus nur Nullen besteht. Als Hinweis dafür kann man durch einem einfachen AND Gatter mit negierten Eingänge eine Fehlermeldung dem Benutzer anzeigen. Zusätzliche Maßnahmen müssen auch getroffen werden, dass der Seed nicht während eines Eins am Clock verändert wird.

Dank seiner Geschwindigkeit und einfachem Aufbau ist das LFSR besonders gut als Bitfehlertester geeignet. Dazu braucht man zwei gleich eingestellte LFSR und zusätzlich, für die Empfängerseite, eine Komparator-Schaltung mit der entsprechende Fehlermeldung.

Wegen dem begrenzten Umfang dieser Arbeit, wurde hier nicht ausführlich über grundlegende Bauteile diskutiert, oder über den mathematischen Rahmen, woraus die Gleichungen und die Tabellen berechnet werden. Um die Abbildungen in überschaubare Größen zu halten, haben wir auch auf breitere Schieberegister verzichtet. Deren Aufbau und Funktion folgt aber die selbe Regel und Schema und kann effektiver auf dem Monitor simuliert und beobachtet werden.

Literaturverzeichnis

KOCH, A. (o.J.)
Boolesche Algebra und kombinatorische Schaltkreise, AKAD-Studienbrief ELT302, o.O.

KOCH, A. et al.(o.J.)
Sequenzielle Schaltungen, Schaltwerke und Simulationssoftware, AKAD-Studienbrief ELT303, o.O.

O.V. (2005)
Linear Feedback Shift Registers,
http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm (Zugriff am 15.07.2018)

KOOPMAN, P. (o.J.)
Maximal Length LFSR Feedback Terms,
https://users.ece.cmu.edu/~koopman/lfsr/index.html (Zugriff am 15.07.2018)

GISSELQUIST, D. (2017)
Generating Pseudo-Random Numbers on an FPGA,
http://zipcpu.com/dsp/2017/10/27/lfsr.html (Zugriff am 15.07.2018)

WARD, R.; MOLTENO, T (2007)
Table of Linear Feedback Shift Registers,
http://courses.cse.tamu.edu/walker/csce680/lfsr_table.pdf (Zugriff am 15.07.2018)

ALFKE, P. (1996)
Efficient Shift Registers, LFSR Counters, and Long Pseudo- Random Sequence Generators,
https://www.xilinx.com/support/documentation/application_notes/xapp052.pdf (Zugriff am 15.07.2018)

LABLANS, P. (2013)
LFSR based scramblers and descramblers,
http://lfsrscrambler.com/ (Zugriff am 15.07.2018)

SALUJA, K. (1991)
Linear Feedback Shift Registers Theory and Applications,
http://homepages.cae.wisc.edu/~ece553/handouts/LFSR-notes.PDF (Zugriff am 15.07.2018)

KOETER, J. (1996)
What's an LFSR?,
http://www.ti.com/lit/an/scta036a/scta036a.pdf (Zugriff am 15.07.2018)

VIEGA, J. (o.J.)
Practical Random Number Generation in Software,
https://www.acsac.org/2003/papers/79.pdf (Zugriff am 15.07.2018)