

# Grundlagen der digitalen Elektronik

**Professor Barry Paton**  
**Physics, Dalhousie University**

Deutsche Übersetzung: Institut für Mechatronik und  
Automatisierungstechnik der Hochschule Rapperswil,  
Schweiz  
ima@hsr.ch (Mai 1999)

## **Copyright**

Copyright © 1998 National Instruments Corporation, 6504 Bridge Point Parkway, Austin, Texas 78730-5039.

Universitäten, Schulen oder andere Bildungszentren dürfen diese Unterlage ganz oder in Teilen für schulischen Gebrauch vervielfältigen. Bei anderem Gebrauch darf diese Unterlage nicht ohne vorherige schriftliche Genehmigung von National Instruments Corporation auf irgendeine Art elektronisch oder mechanisch - einschliesslich Fotokopien, Aufnahmen, Speicherung auf einem Datenwiedergewinnungssystem oder Übersetzungen - ganz oder in Teilen vervielfältigt oder übertragen werden.

## **Warenzeichen**

LabVIEW™ und The Software is the Instrument™ sind Warenzeichen der National Instruments Corporation.

Die aufgelisteten Produkt- und Firmennamen sind Warenzeichen oder Handelsnamen der jeweiligen Unternehmen.

**Für mehr Information**

Falls Sie Fragen oder Anmerkungen betreffend diesem Kurshandbuch haben, gehen Sie bitte auf die folgende Web-Seite: <http://sensor.phys.dal.ca/DigitalElectronics/>

**Internationale Niederlassungen**

Australien 03 9879 5166, Belgien 02 757 00 20, Brasilien 011 288 3336, Dänemark 45 76 26 00, Deutschland 089 741 31 30, Finnland 09 725 725 11, Frankreich 01 48 14 24 24, Grossbritannien 01635 523545, Hongkong 2645 3186, Israel 03 6120092, Italien 02 413091, Japan 03 5472 2970, Kanada (Ontario) 905 785 0085, Kanada (Québec) 514 694 8521, Korea 02 596 7456, Mexiko 5 520 2635, Niederlande 0348 433466, Norwegen 32 84 84 00, Österreich 0662 45 79 90 0, Schweden 08 730 49 70, Schweiz 056 200 51 51, Singapur 2265886, Spanien 91 640 0085, Taiwan 02 377 1200

**National Instruments-Firmenhauptsitz**

6504 Bridge Point Parkway Austin, Texas 78730-5039 Tel.: 512 794 0100

# Inhaltsverzeichnis

## Einleitung

### Übung 1: Gatter

Das AND-Gatter .....	1-1
Das OR- und XOR-Gatter.....	1-2
Inversion .....	1-2
Das NAND-, NOR- und NXOR-Gatter.....	1-2
Erstellung von Gattern mit anderen Gattern .....	1-3
Gatter mit mehr als zwei Eingängen.....	1-4
Maskierung .....	1-5
Anwendung: Datenweiche.....	1-6
Benennung der Gatter .....	1-7
Bibliothek der VIs der Übung 1.....	1-7

### Übung 2: Codierer und Decodierer

Der Würfel.....	2-2
Modulo-6-Zähler.....	2-3
Codierer .....	2-4
Virtueller Würfel.....	2-5
Bibliothek der VIs der Übung 2.....	2-6

### Übung 3: Binäre Addition

Addierer-Erweiterung .....	3-3
Binär codierte Dezimalzahl (BCD).....	3-6
LabVIEW-Knacknuss .....	3-6
Bibliothek der VIs der Übung 3.....	3-7

### Übung 4: Speicherelement: Einzustandsgesteuertes D-Flip-Flop (D-Latch)

Schieberegister (Shift Register).....	4-2
LabVIEW-Knacknuss .....	4-4
Ringzähler.....	4-4
Bibliothek der VIs der Übung 4.....	4-5

### Übung 5: Pseudo-Zufallszahlen-Generator

Ein 6-Bit Pseudo-Zufallszahlen-Generator.....	5-1
Ein 8-Bit Pseudo-Zufallszahlen-Sequenzener .....	5-2
8-Bit Pseudo-Zufallszahlen-Generator .....	5-4
Codierung der digitalen Daten.....	5-5
Bibliothek der VIs der Übung 5.....	5-6

**Übung 6: JK-Master-Slave Flip-Flop**

Binäre Zähler ..... 6-3  
 8-Bit Zähler ..... 6-5  
 LabVIEW-Knacknuss ..... 6-5  
 Zusammenfassung ..... 6-5  
 Bibliothek der VIs der Übung 6 ..... 6-6

**Übung 7: Digital/Analog-Wandler (DAC)**

Was ist ein DAC? ..... 7-1  
 ALU Simulator ..... 7-3  
 Simulation eines echten DAC-Bausteins ..... 7-4  
 Funktionsgenerator ..... 7-5  
 Spezielle DACs ..... 7-6  
 Lissajous Figuren ..... 7-7  
 Bibliothek der VIs der Übung 7 ..... 7-7

**Übung 8: Analog/Digital-Wandler (ADC), Teil I**

Zweck des Analog/Digital-Wandlers ..... 8-1  
 Der Rampen-ADC ..... 8-2  
 LabVIEW-Knacknuss ..... 8-4  
 Tracking ADC ..... 8-4  
 Bibliothek der VIs der Übung 8 ..... 8-7

**Übung 9: Analog/Digital-Wandler (ADC), Teil II**

SAR Simulation ..... 9-3  
 Zusammenfassung ..... 9-4  
 Bibliothek der VIs der Übung 9 ..... 9-4

**Übung 10: 7-Segment-Digitalanzeige**

7-Segment-Anzeige ..... 10-1  
 LabVIEW-Knacknuss ..... 10-5  
 Bibliothek der VIs der Übung 10 ..... 10-6

**Übung 11: Serielle Datenübertragung**

Serielle Sender ..... 11-2  
 Serieller Sender mit Spannungseingang ..... 11-5  
 LabVIEW-Knacknuss ..... 11-7  
 Bibliothek der VIs der Übung 11 ..... 11-7

**Übung 12: Zentraleinheit**

Funktionsweise des Rechenwerks (ALU) ..... 12-2  
 Der Akkumulator ..... 12-3  
 Addition ..... 12-5  
 Binäre Zähler ..... 12-5  
 LabVIEW-Knacknuss ..... 12-6  
 Bibliothek der VIs der Übung 12 ..... 12-7

# Einleitung

Die digitale Elektronik ist eine der grundlegenden Disziplinen der Elektrotechnik. Die grosse Vielzahl der in LabVIEW vorhandenen booleschen und numerischen Controls und Indicators zusammen mit der Fülle der Programmstrukturen und Funktionen macht LabVIEW zu einem ausgezeichneten Werkzeug, um viele der grundlegenden Konzepte der digitalen Elektronik sichtbar machen und zeigen zu können. Die zugehörige Modularität von LabVIEW wird in der Weise ausgenutzt, dass komplizierte integrierte Digitalschaltungen aus einfacheren Schaltungen, welche mit den grundlegenden Gattern errichtet werden, aufgebaut werden können. Dieses Handbuch ist als Hilfsmittel für den Unterricht gedacht, sei es für Demonstrationen im Klassenzimmer, für gemeinsame Studien in Lehrveranstaltungen oder sei es für interaktive Übungen im Labor.

Die Reihenfolge der Übungen folgt den meisten Lehrbüchern. Die ersten sechs Übungen umfassen die Grundlagenschaltungen der Gatter, die Codierer, die binären Addierer, die D-Flip-Flops, die Ringzähler und die JK-Flip-Flops. Viele der virtuellen Instrumente (VIs) sind sowohl für Klassenzimmervorführungen wie auch für die Laborpraktika geeignet.

Die folgenden sechs Übungen umfassen Lektionen über Digital/Analog-Wandler (DAC), Analog/Digital-Wandler (ADC), 7-Segment-Anzeigen, serielle Datenübertragung und die Zentraleinheit (CPU). Die Übungen werden am besten in einem Elektroniklabor durchgeführt, um die LabVIEW-Simulationen mit realen, integrierten Schaltungen vergleichen zu können. Alle beschriebenen Simulationen lassen sich in der Realität einsetzen, indem eine DAQ-Karte von National Instruments verwendet wird und damit mit LabVIEW durch digitale und analoge Ein- und Ausgänge und serielle VIs reale Prozesse beeinflusst werden können.

In den Übungen 2, 5 und 12 werden Codierschemas, die digitale Verschlüsselung und die Operationen einer CPU an spezifischen Anwendungen aufgezeigt und demonstriert. Diese Übungen können als herausfordernde Aufgaben in einer Lehrveranstaltung oder Übungsstunde gestellt werden.

Die Übungen können auch gruppiert werden, um spezielle Abhängigkeiten fortgeschrittener Bausteine von grundlegenden Gattern zu demonstrieren. So ist z.B. der Betrieb der Zentraleinheit vom Konzept der Register und von zwei Eingangsoperationen abhängig.

## Anmerkung der Übersetzer

Im Text werden immer dann englische Begriffe benutzt, wenn uns eine Übersetzung ins Deutsche nicht sinnvoll erschien. Wir denken, dass diese Lösung für den Leser hilfreich ist, da die Literatur über digitale Elektronik überwiegend in Englisch abgefasst ist.

## Begriffe englisch - deutsch

addend	Summand
adder	Addierer
Analog-to-Digital Converter (ADC)	Analog/Digital-Wandler
Arithmetic and Logic Unit (ALU)	Rechenwerk
binary counter	Binärzähler
bucket	Eimer
carry	Übertrag
Central Processing Unit (CPU)	Zentraleinheit
character	Zeichen
chart	Diagramm, Anzeige
circuit	Schaltung
clear	löschen, rücksetzen
clock	<i>hier:</i> Takt
clocked	<i>hier:</i> getaktet
comparator	Komparator
counter	Zähler
cycle	Zyklus
data select	Datenweiche
decoder	Decodierer
dice	Würfel
digit	Ziffer
disallowed	nicht erlaubt
display	Anzeige
Digital-to-Analog Converter (DAC)	Digital/Analog-Wandler
edge-triggered devices	flankengetriggerte Bausteine
encoder	Codierer
even	gerade
feedback	Rückführung
full adder	Volladdierer
function	Funktion
gate	Gatter
half adder	Halbaddierer
initialization	Initialisierung
input level	Eingangssgröße
input voltage	Eingangsspannung
keep	behalten
Least Significant Bit (LSB)	niedrigstwertige Bit
line	<i>hier:</i> Datenleitung
Most Significant Bit (MSB)	höchstwertige Bit
numeric	numerisch
odd	ungerade
output	Ausgabe
previous	vorherig

Pseudo-Random Bit Sequencer (PRBS)	Pseudo-Zufallsbit-Sequenzier
Pseudo-Random Number Generator (PRNG)	Pseudo-Zufallszahlen-Generator
range	Bereich
receive	empfangen
repeat	wiederholen
reset	rücksetzen
resistor	Widerstand
resolution	Auflösung
rotate	Rotation
select	auswählen
serial out	serieller Ausgang
set	setzen
shift register	Schieberegister
signed arithmetic	vorzeichenversehene Arithmetik
successive approximation	stufen- oder schrittweise Annäherung
sum	Summe
switch	Schalter
test level	Vergleichsgrösse
test voltage	Vergleichsspannung
trace	Spur
transmit	übertragen
truth table	Wahrheitstabelle
unsigned arithmetic	vorzeichenlose Arithmetik
value	Wert
voltage	Spannung

# Übung 1 – Gatter

Gatter bilden die grundlegenden Bausteine von digitalen Logikschaltkreisen. Diese Bausteine "öffnen" oder "schliessen" sich, um den Durchgang eines logischen Signals zuzulassen oder zu verhindern. Aus nur einer Hand voll grundlegender Gatterarten (AND, OR, XOR und NOT) kann eine beträchtliche Anzahl Gatterfunktionen erstellt werden.

## Das AND-Gatter

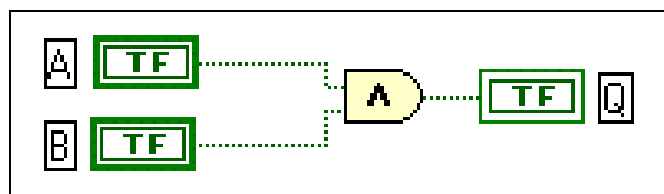
Ein AND-Gatter besteht aus zwei Eingängen, A und B, und einem Ausgang, häufig mit Q bezeichnet. Beim AND-Gatter ist der Ausgang nur "ON", wenn A *und* B auch "ON" sind.

In der digitalen Elektronik wird der "ON"-Zustand häufig durch 1 und der "OFF"-Zustand durch 0 dargestellt. Der Zusammenhang zwischen den Eingangssignalen und den Ausgangssignalen wird oftmals in einer Wahrheitstabelle zusammengefasst, die eine Tabellierung aller möglichen Eingänge und der resultierenden Ausgänge ist. Für AND-Gatter gibt es vier mögliche Eingangs-Kombinationen: A=0, B=0; A=0, B=1; A=1, B=0 und A=1, B=1. In der folgenden Wahrheitstabelle werden diese in der linken und mittleren Spalte aufgelistet. Der Gatterausgang wird in der rechten Spalte tabelliert.

<i>A</i>	<i>B</i>	<i>Q=A AND B</i>
0	0	0
0	1	0
1	0	0
1	1	1

**Tabelle 1-1.** Wahrheitstabelle eines AND-Gatters

In LabVIEW können Sie ein digitales Eingangssignal durch das Umschalten eines Booleschen Schalters eingeben; eine Boolesche LED-Anzeige kann einen Ausgangszustand anzeigen. Da das AND-Gatter in LabVIEW bereits als Basisfunktion zur Verfügung steht, können Sie auf einfache Weise ein VI erzeugen, das eine AND-Verknüpfung darstellt: Verdrahten Sie zwei Schalter zu den Gattereingängen und eine LED-Anzeige zum Ausgang.

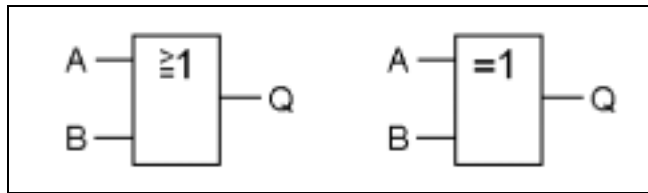


**Abbildung 1-1.** Verdrahtung der Ein- und Ausgänge der AND-Funktion in LabVIEW

Starten Sie **AND gate.vi** von der VI-Bibliothek **Chap 1.llb** und betätigen Sie die zwei Eingangstasten. Beobachten Sie dabei, wie sich der Ausgang ändert und überprüfen Sie die oben abgebildete Wahrheitstabelle.

### Das OR- und XOR-Gatter

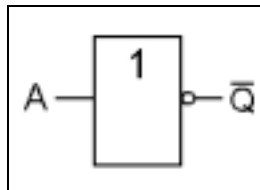
Das OR-Gatter besteht ebenfalls aus zwei Eingängen und einem Ausgang. Anders als beim AND-Gatter ist der Ausgang dann 1, wenn einer der beiden Eingänge 1 ist, oder aber, wenn beide Eingänge 1 sind. Der Gatterausgang ist nur dann 0, wenn beide Eingänge 0 sind.



**Abbildung 1-2.** Schaltzeichen für das OR- und XOR-Gatter

Ein dem OR-Gatter ähnliches Gatter ist das XOR, oder das eXklusive **OR**, bei welchem der Ausgang 1 ist, wenn *nur einer* der Eingänge 1 ist. Mit anderen Worten ist der XOR-Ausgang 1, wenn die Eingänge verschieden sind.

### Inversion

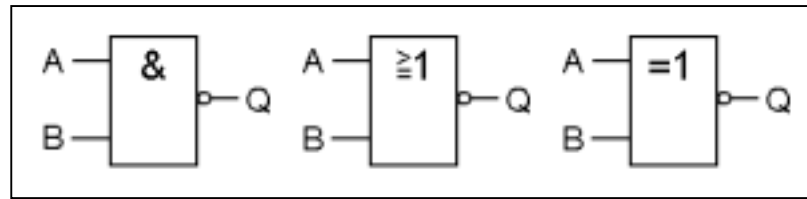


**Abbildung 1-3.** Das NOT-Gatter

Das NOT-Gatter ist eine einfache Verknüpfung mit nur einem Eingang und einem Ausgang. Der Ausgang ist jeweils vom Eingang verschieden bzw. invertiert.

### Das NAND-, NOR- und NXOR-Gatter

Die Inversion ist ziemlich hilfreich. Zu den bereits besprochenen drei Gattern mit ihren zwei Eingängen gibt es zusätzlich noch drei weitere. Diese sind mit den AND-, OR- und XOR-Gattern identisch, ausser dass der Gatterausgang invertiert ist. Sie werden NAND ("not AND"), NOR ("not OR") und NXOR ("not exclusive OR") - Gatter genannt. Ihre Schaltzeichen sind dieselben wie die der nicht invertierten Gatter, weisen jedoch am Ausgang einen kleinen Kreis auf.



**Abbildung 1-4.** Invertiertes AND-, OR- und XOR-Gatter

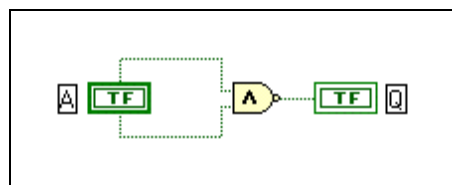
Starten Sie **Truth table.vi** und wählen Sie eines der Gatter aus. Führen Sie nun alle möglichen Kombinationen durch, um so die folgende Wahrheitstabelle zu vervollständigen.

A	B	AND	OR	XOR	NAND	NOR	NXOR
0	0	0					
0	1	0					
1	0	0					
1	1	1					

**Tabelle 1-2.** Wahrheitstabelle

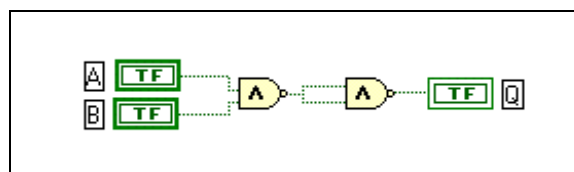
### Erstellung von Gattern mit anderen Gattern

Mit ein paar wenigen NAND-Gattern können Sie alle weiteren Basisgatter reproduzieren. Z.B. können Sie ein NOT-Gatter bilden, indem Sie die beiden Eingänge eines NAND-Gatters mit dem gleichen Eingang verbinden.



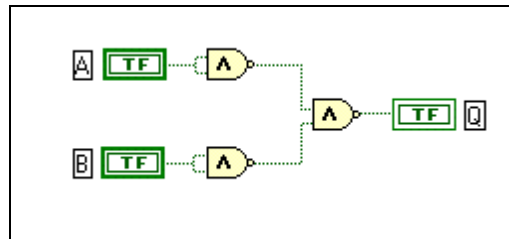
**Abbildung 1-5.** NOT-Gatter gebildet mit einem NAND-Gatter

Auf die gleiche Weise können Sie ein AND-Gatter mit zwei NAND-Gattern bilden:



**Abbildung 1-6.** AND-Gatter gebildet mit zwei NAND-Gattern

Ein OR erfordert drei NAND-Gatter:



**Abbildung 1-7.** OR-Gatter gebildet mit drei NAND-Gattern

Erstellen Sie ein VI, welches zeigt, daß ein XOR-Gatter aus vier NAND-Gattern gebildet werden kann. Als Referenz können Sie in der Bibliothek der Übung 1 unter **XOR from NAND.vi** nachsehen.

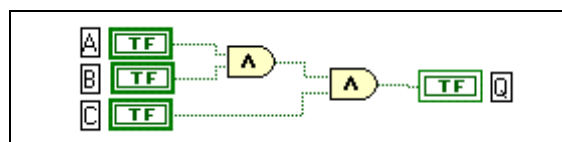
### Gatter mit mehr als zwei Eingängen

Obgleich LabVIEW alle Basisgatter mit zwei Eingängen umfaßt, kann es vorkommen, dass Sie mehr Eingänge benötigen. So kann z.B. die unter Tabelle 1-1 aufgeführte Wahrheitstabelle für das AND-Gatter auf drei Eingänge erweitert werden:

A	B	C	A AND B AND C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

**Tabelle 1-3.** Wahrheitstabelle für ein AND-Gatter mit drei Eingängen

Mit zwei AND-Gattern mit jeweils zwei Eingängen können Sie leicht ein AND-Gatter mit drei Eingängen aufbauen:



**Abbildung 1-8.** LabVIEW-Programm eines AND-Gatters mit drei Eingängen

Öffnen Sie das VI **3 AND.vi** und beachten Sie den Rahmen und das Symbol, welche das VI zu einem SubVI machen.

### Maskierung

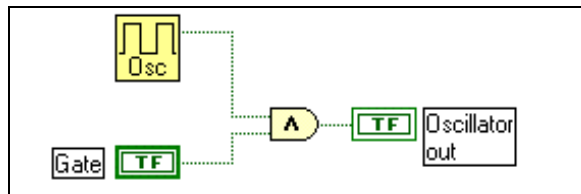
Als einfache Anwendung, wie diese Gatter kombiniert werden können, betrachten wir die *Maskierung*. Um dies zu veranschaulichen, ist unten die Wahrheitstabelle mit den Spaltenüberschriften abgebildet.

A	Maske	A AND B	Resultat	
0	0	0	A ist blockiert	Gatter ist geschlossen
1	0	0		
0	1	0	A ist unverändert	Gatter ist offen
1	1	1		

**Tabelle 1-4.** Wahrheitstabelle für ein AND-Gatter mit einem Eingang als Maske

Wie die Wahrheitstabelle zeigt, kann das AND-Gatter als elektronischer Schalter benutzt werden.

Dieser Sachverhalt wird auf einfache Weise in LabVIEW demonstriert:



**Abbildung 1-9.** AND-Gatter als elektronischer Schalter

**E-switch.vi** zeigt die Arbeitsweise des elektronischen Schalters. Die Wahrheitstabellen lassen sich auch für andere Gatter mit der Maskierung erstellen. In der folgenden Tabelle bedeutet zurückgesetzt "erzwungen zu 0", während gesetzt "erzwungen zu 1" bedeutet:

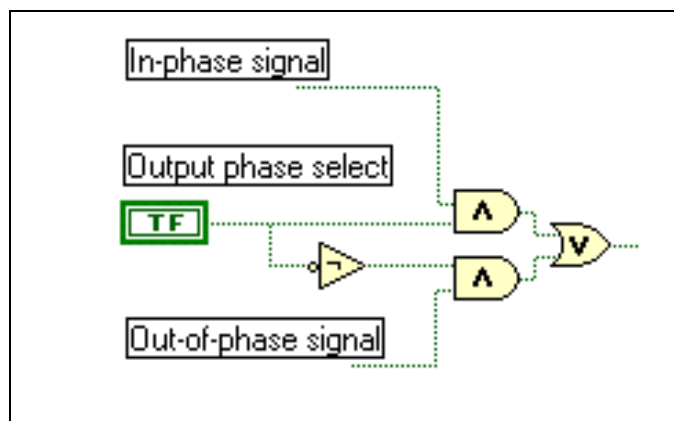
A	Maske	AND	OR	XOR
0	0	A ist zurückgesetzt	A ist unverändert	A ist unverändert
1	0			
0	1	A ist unverändert	A ist gesetzt	A ist invertiert
1	1			

**Tabelle 1-5.** Wahrheitstabelle für ein AND-, OR- und XOR- Gatter mit einem Eingang als Maske

Zusammenfassend sehen wir hier drei hilfreiche Funktionen. Um einen Zustand zu setzen, benutzt man ein OR-Gatter mit einer Maske von 1. Um einen Zustand zurückzusetzen, muss man ein AND-Gatter mit einer Maske von 0 gebrauchen. Will man einen Zustand invertieren, so verwendet man ein XOR-Gatter mit einer Maske von 1.

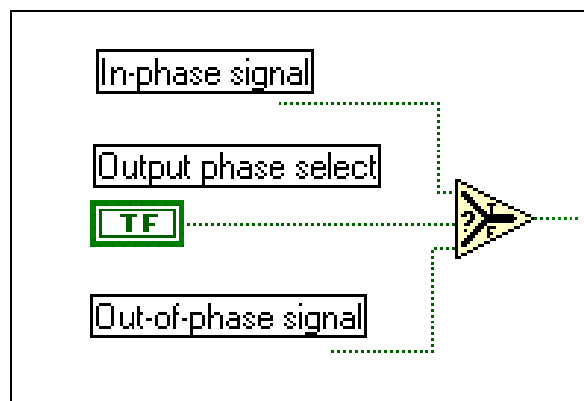
**Anwendung: Datenweiche**

Eine weitere einfache Anwendung der Basisgatter ist die Datenweiche, in der ein einzelner digitaler Eingang eines von zwei digitalen Signalen auswählt:



**Abbildung 1-10.** Digitale Datenweiche gebildet aus den Basisgattern

LabVIEW umfasst eine eingebaute Funktion, **Select** genannt, mit welcher diese Operation nachgebildet werden kann. Die Datenweiche in obiger Abbildung kann damit auf folgende Art und Weise gebildet werden:



**Abbildung 1-11.** Datenweiche realisiert mit LabVIEW

## Benennung der Gatter

Die in dieser Übung vorgestellten Gatter bilden die Grundlage der digitalen Elektronik. Die Vertrautheit mit den Wahrheitstabellen ist sehr nützlich. Repetieren und testen Sie ihr Wissen mit **Name that Gate.vi**.

## Bibliothek der VIs der Übung 1 (in der gleichen Reihenfolge wie bearbeitet aufgelistet)

**AND gate.vi** (AND-Verknüpfung mit zwei Eingängen)

**Truth table.vi** (Wahrheitstabelle für AND, OR, XOR, NAND, NOR und NXOR)

**XOR from NAND.vi**

**3 AND.vi** (AND-Verknüpfung mit drei Eingängen)

**Masking.vi** (Demonstration)

**E-switch.vi** (elektronischer Schalter)

**Data select.vi** (Datenweiche gebildet mit Basisgattern)

**Data select2.vi** (Datenweiche mit der LabVIEW Funktion **Select** erzeugt)

**Oscillator.vi** (SubVI, welches in **Data select.vi** verwendet wurde)

**Name that gate.vi** (Testen Sie Ihr Wissen!)

## Übung 2 - Codierer und Decodierer

Ein Codierer wandelt den Zustand eines Einganges in eine binäre Darstellung um. Betrachten wir dazu einen Drehschalter mit 10 Positionen für die Eingabe der Zahlen von 0 bis 9. Jede Schalterstellung ist nun in eine eindeutige binäre Reihenfolge umzusetzen. Zum Beispiel könnte die Schalterstellung 7 als 0111 codiert werden. Ein Decodierer führt die gegenteilige Konvertierung von den binären Codes in Ausgangscodes durch.

Betrachten wir einen einzelnen Würfel. Auf jeder seiner sechs Seiten erscheint eines der folgenden Muster, welche die Zahlen 1-6 darstellen.

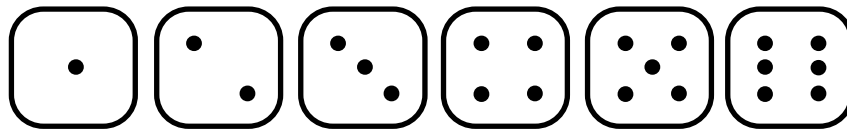


Abbildung 2-1. Die sechs Würfelseiten

Diese Muster sind üblich. Man denke sich nun sieben zu einem "H" angeordnete Lampen:

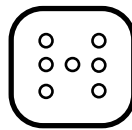


Abbildung 2-2. Punktanordnung für die Würfelmuster

Indem Sie die passenden Lampen einschalten, können Sie jedes der sechs Würfelmuster erzeugen.

Nach näherer Betrachtung erkennt man, dass es vier einzigartige Muster gibt, mit denen jedes der sechs Würfelmuster gebildet werden kann. Wir nennen diese vier Muster Basismuster A, B, C und D:

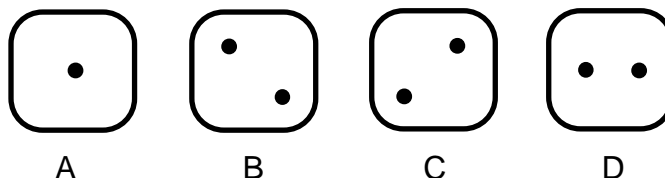


Abbildung 2-3. Die vier Basismuster für die Bildung der Würfelmuster

Hält man in einer Tabelle fest, aus welchen Basismustern die sechs Würfelzahlen entstehen, so wird die Bedeutung dieser vier Basismuster klar.

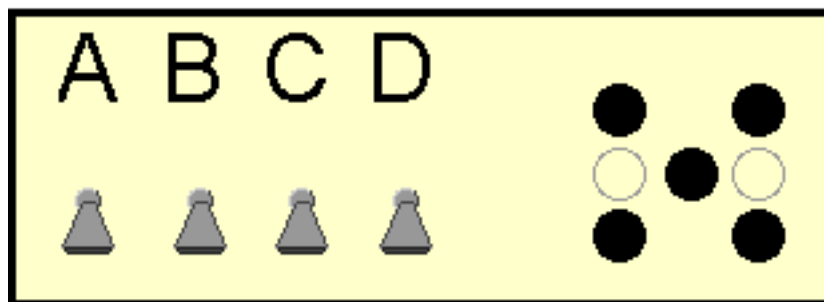
Würfelzahlen	A	B	C	D
1	√			
2		√		
3	√	√		
4		√	√	
5	√	√	√	
6		√	√	√

**Tabelle 2-1.** Basismuster für die Bildung jeder Würfelzahl

Das Basismuster A wird für alle ungeraden Zahlen 1, 3 und 5 benutzt. Basismuster B kommt in der Darstellung von allen Zahlen vor, ausgenommen bei der Zahl 1. Basismuster C tritt bei den Zahlen 4, 5 und 6 auf und Muster D wird nur für die Darstellung der Zahl 6 verwendet.

### Der Würfel

Um einen virtuellen Würfel zu erstellen, benötigt man sieben LED-Anzeigen und vier Schalter. Im Front Panel sind die LED-Anzeigen zu einem "H" anzuordnen und im Diagrammfenster müssen die LEDs verdrahtet werden, um die vier Basismuster A, B, C und D anzeigen zu können. Durch Betätigen der vier Kippschalter im Front Panel können nun die Basismuster erzeugt werden.



**Abbildung 2-4.** LabVIEW-Front Panel für den virtuellen Würfel

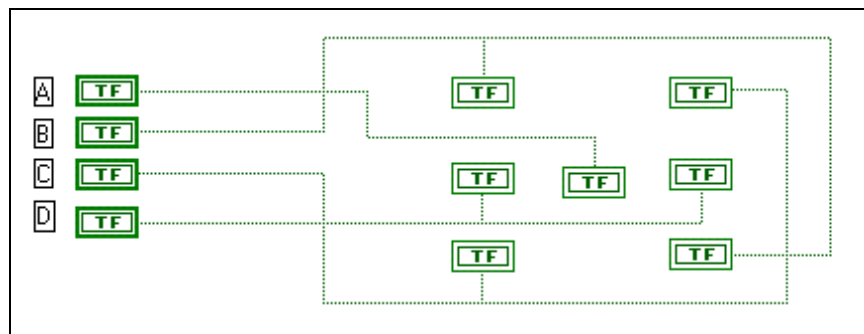


Abbildung 2-5. LabVIEW-Blockdiagramm für die Erzeugung des virtuellen Würfels

Laden Sie das VI **Display.vi** und sehen Sie sich die Funktion des virtuellen Würfels an.

### Modulo-6-Zähler

Ein Modulo-6-Zähler ist ein Zähler mit sechs eindeutigen Zuständen, die sich in der gleichen Reihenfolge wiederholen. Sie können einen einfachen Modulo-6-Zähler bilden, indem Sie ein Schieberegister (Shift Register) mit drei Elementen am linken Rand benutzen und dabei den dritten Elementausgang invertieren und zum rechten Element führen. (Solch ein Zähler wird auch "switched tail ring counter" genannt).

Öffnen Sie ein neues LabVIEW VI und platzieren Sie drei LED-Anzeigen in das Front Panel. Diese zeigen den Ausgangszustand der Schieberegisterelemente Q1, Q2 und Q3, an. Fügen Sie dem Blockdiagramm ein Schieberegister mit drei Elementen hinzu und verbinden Sie jedes dieser Elemente mit einer LED-Anzeige. Sie können eine Wartezeitfunktion verwenden, um den Vorgang für eine Demonstration zu verlangsamen. Beachten Sie, dass die While-Schleife links nicht angeschlossen ist. Bei jedem Durchgang dieses VIs wird der nächste Wert zurückgegeben. Selektieren Sie im Front Panel die drei Ausgänge als Verbindung im Icon Editor und sichern Sie dieses Programm als SubVI unter dem Namen **Rotate.vi**.

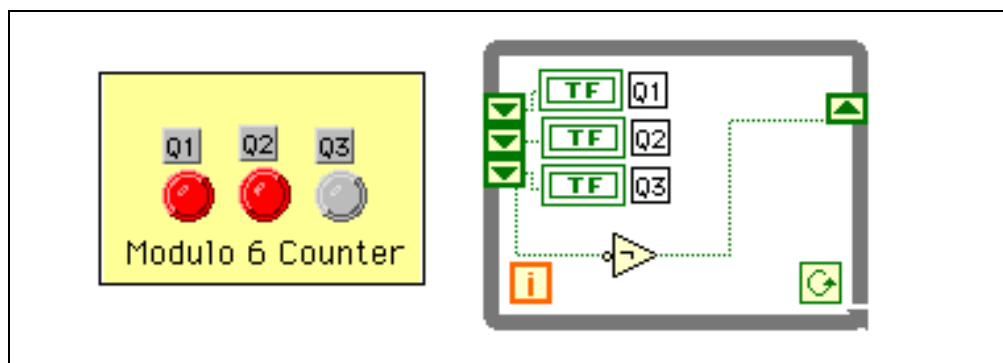


Abbildung 2-6. Front Panel und Blockdiagramm von **Rotate.vi**

Unten ist die Wahrheitstabelle für den Modulo-6-Zähler abgebildet. Lassen Sie das Programm siebenmal laufen und beobachten Sie dessen Funktion.

Zyklus	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>
1	0	0	0
2	1	0	0
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	0	0	0

wie Zyklus 1

**Tabelle 2-2.** Wahrheitstabelle für einen Modulo-6-Zähler

Die Ausgangswerte wiederholen sich nach sechs Zählimpulsen. Daher kommt der Name Modulo-6-Zähler.

**Codierer**

Es gibt keinen Grund, entscheiden zu müssen, welcher Ausgang mit welchem Zählimpuls korrespondiert. Jedoch verbessert eine systematische Darstellung die Übersicht.

#	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>1</sub> '	Q <sub>2</sub> '	Q <sub>3</sub> '
6	0	0	0	1	1	1
4	1	0	0	0	1	1
2	1	1	0	0	0	1
1	1	1	1	0	0	0
3	0	1	1	1	0	0
5	0	0	1	1	1	0

**Tabelle 2-3.** Codierschema für den digitalen Würfel

Zum Beispiel liefert jeder Ausgang drei Einsen und drei Nullen. Einer dieser Ausgänge, z.B. Q<sub>3</sub>, könnte die ungeraden Zahlen 1, 3 und 5 darstellen. Ein anderer Ausgang, z.B. Q<sub>2</sub>', könnte die Zahlen 4, 5, 6 repräsentieren. Diese zwei Kolonnen decodieren dann zwei der vier Basismuster für "frei". Die anderen zwei Basismuster werden mit einem bestimmten Muster der drei Zähler decodiert. Zu diesem Zweck kann ein AND-Gatter mit drei Eingängen, wie wir es in Übung 1 behandelt haben, zusammen mit einem Inverter benutzt werden. "1" (Basismuster A) wird decodiert mit der Kombination Q<sub>1</sub> & Q<sub>2</sub> & Q<sub>3</sub> und die Zahl "6" wird decodiert mit Q<sub>1</sub>' & Q<sub>2</sub>' & Q<sub>3</sub>'.

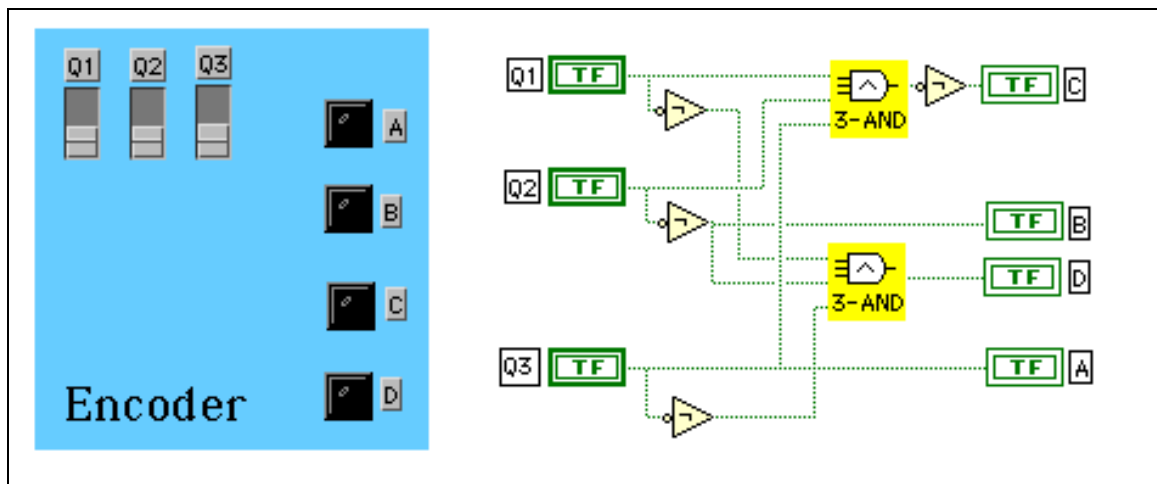


Abbildung 2-7. Front Panel und Blockdiagramm von **Encode.vi**

Der Codierer wird konstruiert, indem man drei Boolesche Anzeigen zusammen mit vier LED-Anzeigen in das Front Panel setzt. Verdrahtet wird der Codierer wie im letzten Abschnitt beschrieben.

### Virtueller Würfel

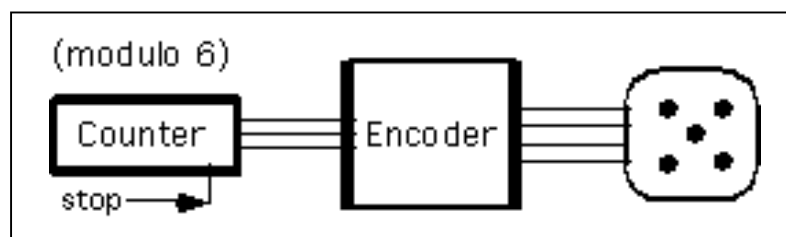


Abbildung 2-8. Funktionsdiagramm des digitalen Würfels

Das Würfeln mit dem virtuellen Würfel wird mit Hilfe eines schnellen Zählers, der alle sechs Würfelzahlen durchläuft, realisiert. Die Würfelzahlen werden an drei Ausgängen codiert. In der Praxis läuft der Zähler, bis ein Stoppbefehl ausgegeben wird. Der Wert zum Zeitpunkt des Stopps entspricht dann dem "gewürfelten Wert". Ein Taktgeber mit einer Frequenz grösser als 1 kHz gewährleistet den Zufallscharakter.

Ein Codierer wandelt nun die drei Signale des Zählers in die vier Steuersignale für die Basismuster um. Diese stellen der Reihe nach die Punkte auf dem virtuellen Würfel mit den korrekten Ausgangscodes dar.

Es ist jetzt noch ein Einfaches, all die einzelnen Funktionen – Zähler, Codierer und Anzeige – zu einem VI, **Dice.vi** genannt, zusammenzubauen. Genauso wie Sie elektronische Stromkreise aus einzelnen Gattern, Verriegelungen, Schaltern und Anzeigen zusammenbauen würden, bildet LabVIEW komplizierte Funktionen aus einfacheren.

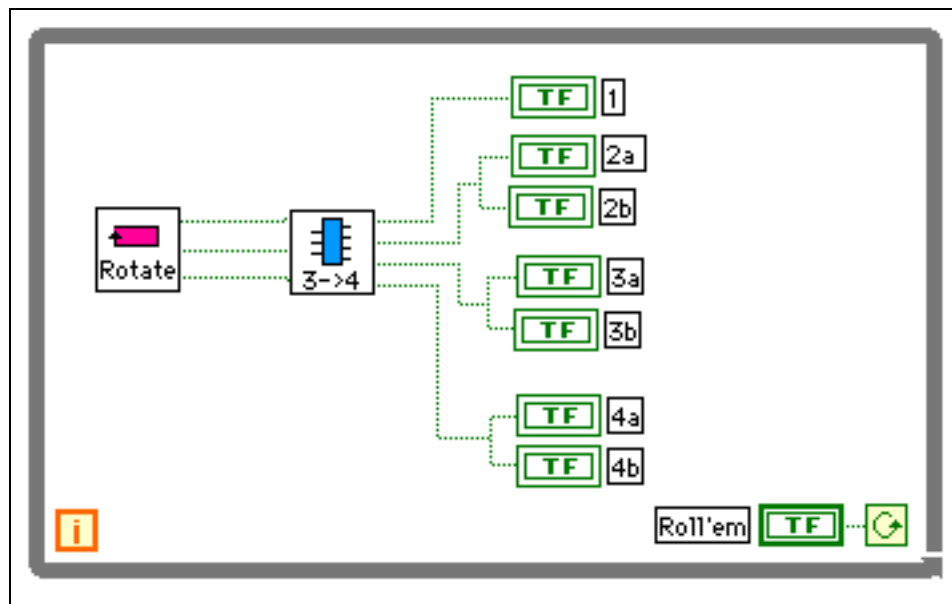


Abbildung 2-9. Blockdiagramm des **Dice.vi**. Beachten Sie die Ähnlichkeit mit dem obigen Funktionsdiagramm

Betätigen Sie jetzt den Schalter im Front Panel und werfen Sie den Würfel!

**Bibliothek der VIs der Übung 2 (in der gleichen Reihenfolge wie bearbeitet aufgelistet)**

**Display.vi** (LED-Anzeige für den virtuellen Würfel)

**Rotate.vi** (Modulo-6-Zähler)

**Encoder.vi** (codiert Zählercodes zu Anzeigecodes)

**3 AND.vi** (SubVI, welches in Encoder.vi verwendet wurde)

**Dice.vi** ("Werfen Sie den Würfel!")

## Übung 3 - Binäre Addition

Bevor wir mit dieser Übung fortfahren, ist es sinnvoll, einige Einzelheiten der binären Addition zu wiederholen. Fügt man bei der binären, wie auch bei der dezimalen Addition irgendeinem Wert 0 hinzu, so bleibt der Wert unverändert:  $0 + 0 = 0$ ,  $1 + 0 = 1$ . Wenn man jedoch  $1 + 1$  in der binären Addition ausführt, ist das Resultat nicht wie bei der dezimalen Addition 2 (ein Symbol, welches im Binärsystem nicht besteht), sondern "10"; eine Eins im "Zweierplatz" und eine Null im "Einerplatz". Schreibt man diese Addition vertikal auf, kann man folgendes dazu sagen: „Eins und eins gleich zwei; schreibe Null, behalte Eins“.

$$\begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$$

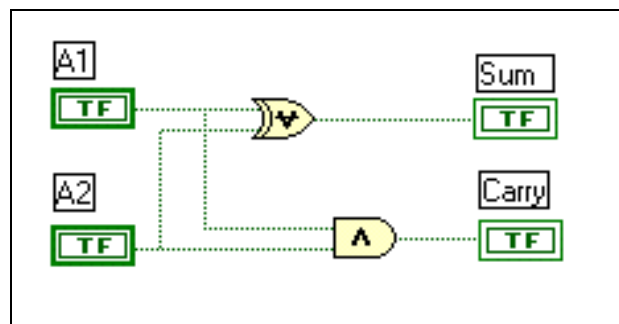
**Abbildung 3-1.** Addition einstelliger Binärzahlen

Unten abgebildet ist die Wertetabelle für die Addition zweier einstelliger Binärzahlen. Dabei gibt es zwei Kolonnen für die Eingänge, eine für jeden Summanden, A1 und A2, und zwei Kolonnen für die Ausgänge, eine für die Summe und eine für den Übertrag:

A1 + A2		= Summe mit Übertrag	
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

**Tabelle 3-1.** Wertetabelle für die Addition

Welches der Gatter können Sie verwenden, um die Resultate der binären Addition zu erhalten? Beachten Sie, dass  $A1 \text{ XOR } A2$  die Ausgabe der Summe ergibt und  $A1 \text{ AND } A2$  die Ausgabe des Übertrags. Mit LabVIEW sieht diese 1-bit Addition wie folgt aus:



**Abbildung 3-2.** Halbaddierer mit einem XOR- und einem AND-Gatter gebildet

Dieser digitale Baustein wird *Halbaddierer* genannt. Die Bezeichnung Halbaddierer bezieht sich auf die Tatsache, dass diese Konfiguration zwar einen Übertrag für ein Bit höherer Ordnung anzeigen kann, jedoch einen Übertrag von einer nächst niedrigeren Stelle nicht akzeptieren kann.

Ein *Volladdierer* hat demnach drei Eingänge, zusätzlich zu den zwei Summanden noch einen "carry in"-Eingang (carry hier: Übertrag). Dieser addiert das Bit, welches von der nächst niedrigeren Stelle übertragen wird, wie dies im folgenden Beispiel in der mittleren Kolonne zu sehen ist:

$$\begin{array}{r} 101 \\ +001 \\ \hline 110 \end{array}$$

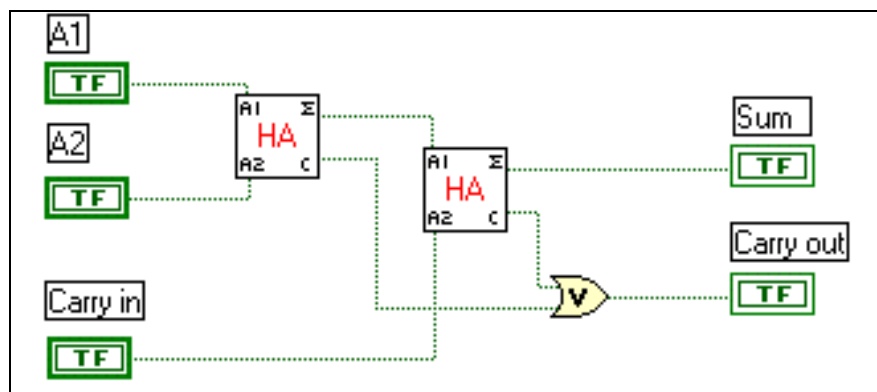
**Abbildung 3-3.** Addition zweier dreistelliger Binärzahlen

Die Wertetabelle für einen Volladdierer weist deshalb drei Eingänge auf und so gibt es acht mögliche Zustände:

Carry in	A1	A2	Summe	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

**Tabelle 3-2.** Wertetabelle für die Addition mit einem "carry in"-Eingang

Beachten Sie, daß alle drei Eingänge im Wesentlichen äquivalent sind; der Volladdierer addiert einfach die drei Eingänge. Eine Möglichkeit, einen Volladdierer zu entwerfen, ist, zwei Halbaddierer zu kombinieren:



**Abbildung 3-4.** Zwei Halbaddierer-SubVIs zu einem Volladdierer kombiniert

Beachten Sie die Einfachheit der Verdrahtung bei Verwendung der zwei Halbaddierer.

### Addierer-Erweiterung

Durch Kombination von 1-Bit-Addierern können Sie die Addition von mehrstelligen Binärzahlen erreichen. Wie im folgenden Beispiel gezeigt wird, führt jeder 1-Bit-Addierer die Addition für eine Kolonne durch:

$$\begin{array}{r}
 1011 \\
 +0010 \\
 \hline
 1101
 \end{array}$$

Abbildung 3-5. Addition vierstelliger Binärzahlen (11+2=13)

Die Addition zweier vierstelliger Binärzahlen könnte mit LabVIEW wie folgt aussehen:

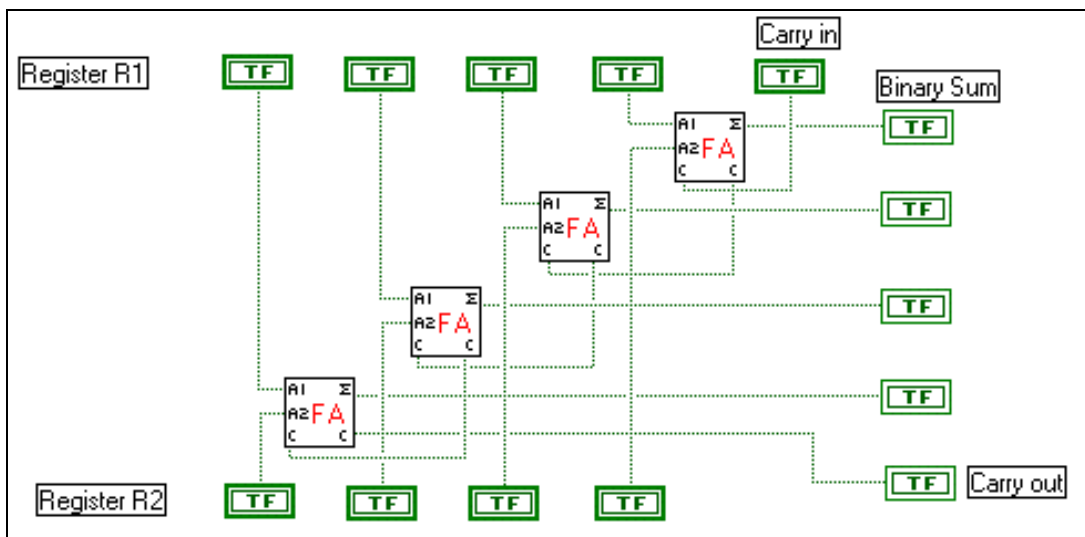
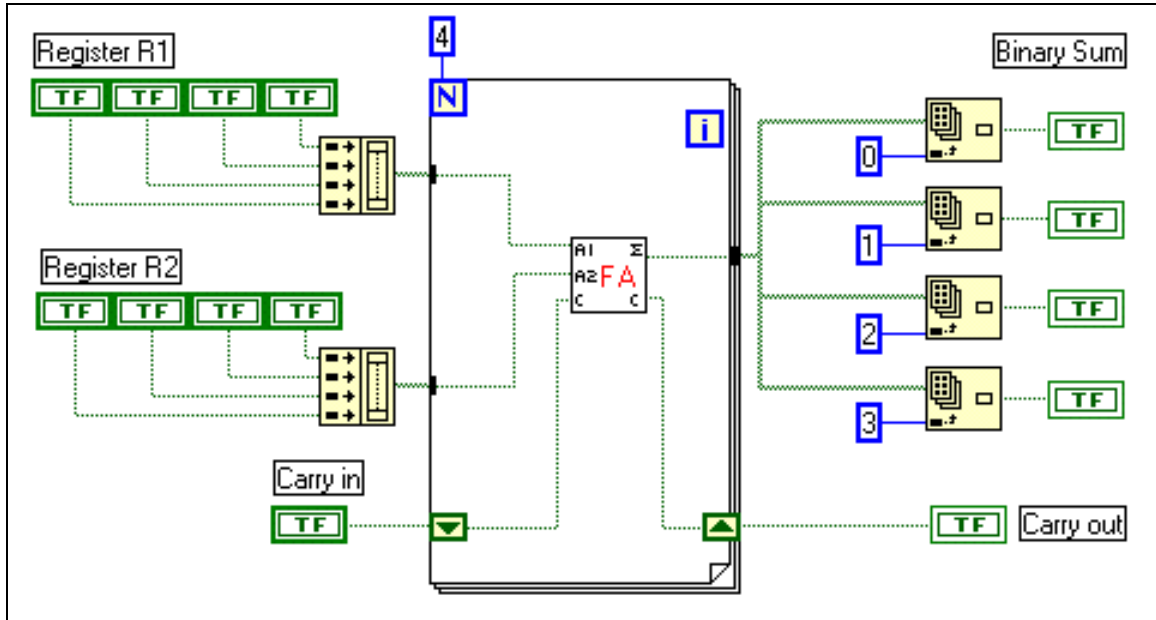


Abbildung 3-6. LabVIEW-Blockdiagramm für die Addition vierstelliger Binärzahlen

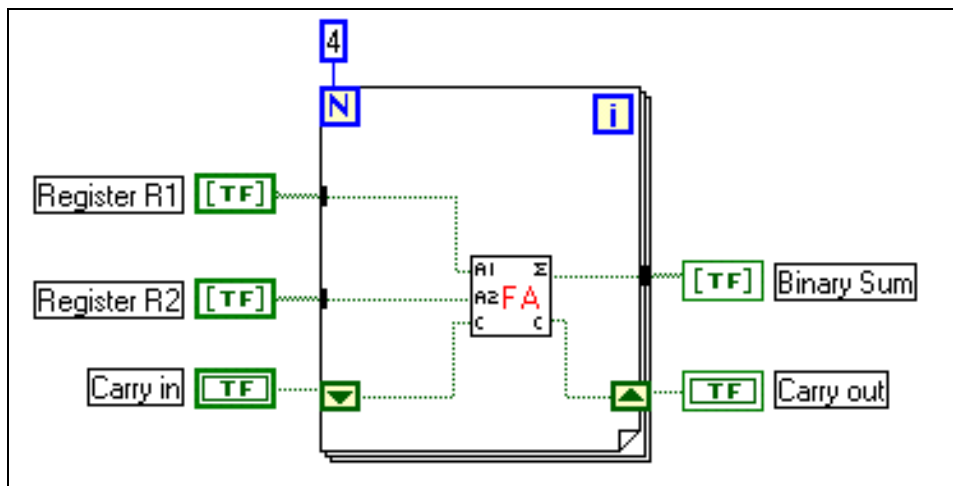
Beachten Sie, dass dieses VI aus vier Volladdierern besteht. Wollen Sie mit dieser Schaltung nur vierstellige Binärzahlen addieren, könnte für die Addition der niedrigsten Stelle ein Halbaddierer eingesetzt werden. Benutzt man jedoch für jedes Additionsmitglied einen Volladdierer, so kann neben den zwei Eingängen für die vierstelligen Summanden noch ein "carry in"-Eingang hinzugenommen werden. Öffnen Sie das VI **Four-bit Adder1.vi** und führen Sie die Addition zweier vierstelliger Binärzahlen aus. Es werden dabei zwei SubVIs verwendet, **Full Adder.vi**, gezeigt in Tabelle 3-4, und **Half Adder.vi**, gezeigt in Tabelle 3-2.

Wie Sie sehen können, ist die Schaltung in Abbildung 3-6 ein wenig komplex und würde noch komplizierter werden, wenn Sie Summanden mit noch mehr Bits addieren wollten. Indem Sie eine For-Schleife mit einem Schieberegister verwenden, können Sie die Schaltung erheblich vereinfachen:



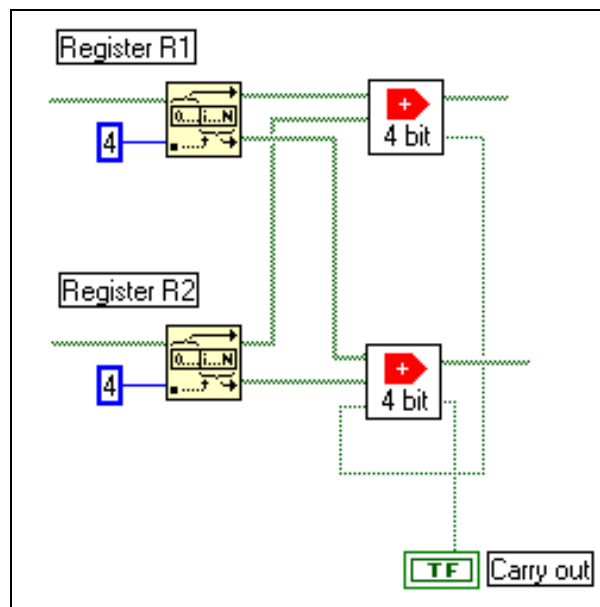
**Abbildung 3-7.** Addition vierstelliger Binärzahlen mit LabVIEW-Arrays realisiert (**Four-Bit Adder2.vi**)

Beachten Sie, dass aus den vier unabhängigen Bits, 4-Bit-Arrays gebildet werden, bevor sie der For-Schleife zugeführt werden. Die For-Schleife addiert bei jedem der vier Wiederholungen ein Bit-Paar, beginnend mit der wertniedrigsten Stelle. Bei der ersten Wiederholung wird der "carry in"-Eingang des Volladdierers vom Front Panel genommen; bei den nachfolgenden Wiederholungen kommt der "carry in"-Eingang von den vorangehenden Schritten. Führen Sie eine Addition mit beiden VI-Varianten durch und überprüfen Sie, dass deren Resultate indentisch sind.



**Abbildung 3-8.** Addition vierstelliger Binärzahlen mit Arrays für die Ein- und Ausgänge

Es gibt noch eine dritte Variante der oben genannten VIs, **Four-bit Adder3.vi** genannt. Diese ist identisch zu der in Abbildung 3-7 gezeigten Variante, ausser daß die Ein- und Ausgänge als Boolesche Arrays dargestellt werden. Dabei ist zu beachten, dass bei Booleschen Arrays das niedrigstwertige Bit (Least Significant Bit, LSB) links und das höchstwertige Bit (Most Significant Bit, MSB) rechts ist. Diese Variante ist als SubVI konfiguriert worden, wodurch Sie aus zwei dieser SubVIs eine Addition achtstelliger Binärzahlen durchführen können. Dabei wird jeder 8-Bit Summand (ein Byte) in zwei 4-Bit Blöcke getrennt, die zwei niedrigstwertigen Blöcke werden dann zum einen 4-Bit Addierer verbunden, die zwei höchstwertigen Blöcke zum anderen 4-Bit Addierer.



**Abbildung 3-9.** 8-Bit Addierer aufgebaut aus zwei 4-Bit Addierern

### Binär codierte Dezimalzahl (BCD)

Bei der digitalen Datenverarbeitung werden Dezimalziffern nicht immer ins Binärsystem umgewandelt. Die Darstellung von binär codierten Dezimalzahlen (BCD) wird ebenfalls verwendet. Beim BCD-Code wird jede Dezimalziffer folgendermassen in vier Bits übersetzt:

Dezimalziffer	BCD-Code	Dezimalziffer	BCD-Code
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

**Tabelle 3-3.** BCD-Darstellung für die Ziffern 0 bis 9

Die BCD-Darstellung kann als Teil der vollen Binärdarstellung betrachtet werden, bei der nur die Bitmuster 0000 bis 1001 (0 bis 9) verwendet werden. Z.B.:

$$42_{10} = 0100\ 0010_{BCD}$$

Beachten Sie, dass sich diese Darstellung eindeutig von der Binärdarstellung unterscheidet. Das Beispiel sieht im Binärsystem dargestellt folgendermassen aus:

$$42_{10} = 0010\ 1010_2$$

Deutlich ist, dass mit dem BCD-Code Bits verschwendet werden, da einige der 4-Bit Muster nicht für die Codierung der Dezimalziffern gebraucht werden. Bei der Darstellung grosser Zahlen wird die Verschwendung der Bits umso deutlicher. Mit zwei Bytes (16 Bits) können mit der binären Darstellung Ganzzahlen im Bereich von 0 bis 65535 dargestellt werden. Verwendet man jedoch den BCD-Code, können mit zwei Bytes nur die Zahlen von 0 bis 9999 dargestellt werden. Der Vorteil des BCD-Codes liegt darin, daß er für die Zifferndarstellung auf Anzeigen bestens geeignet ist.

### LabVIEW-Knacknuss

Entwerfen Sie einen BCD-Codierer, welcher die Zahlen 0 bis 9 als BCD-Code darstellt, wie auch einen BCD-Decodierer, der das Gegenteilige ausführt. Erstellen Sie zudem einen 1-Ziffer-BCD-Addierer.

**Bibliothek der VIs der Übung 3 (in der gleichen Reihenfolge wie bearbeitet aufgelistet)**

**Half Adder.vi** (1-Bit Addition)

**Full Adder.vi** (1-Bit Addition mit „carry in“-Eingang)

**Four-bit Adder1.vi** (addiert zwei 4-Bit Zahlen mit „carry in“-Eingang)

**Four-bit Adder2.vi** (vereinfachte Version)

**Four-bit Adder3.vi** (benutzt Boolesche Arrays für die Ein- und Ausgänge)

**Eight-bit Adder.vi** (benutzt zwei 4-bit Addierer)

## Übung 4 - Speicherelement: Einzustandsgesteuertes D-Flip-Flop (D-Latch)

In den vorangegangenen drei Übungen hatten Sie es mit kombinatorischen Schaltungen zu tun, bei denen sich die Ausgangszustände vollständig aus den Eingangszuständen ergaben. Bis anhin war bei den Schaltungen ein Folgezustand vom ursprünglichen Zustand völlig unabhängig. Auch war egal, auf welchem Wege man einen Zustand erreichte. Das heisst, daß ein Speichern mit diesen Schaltungen nicht möglich war. Die meisten digitalen Operationen laufen sequentiell ab: Ereignis B tritt erst nach Ereignis A auf. Zudem sind die Ereignisse in einem Digitalrechner nicht nur sequentiell, sondern auch synchron mit irgendeinem externen Taktgeber. Getaktete Digitalbausteine sind Bausteine, deren Ausgänge nur ändern, wenn ein Taktgebersignal vorliegt. In den nächsten Übungen sehen Sie, wie durch getaktete Digitalbausteine Speicherfunktionen möglich sind und wie dadurch viele interessante Digitalschaltungen entworfen werden können.

Ein einfaches Speicherelement ist das einzustandsgesteuerte D-Flip-Flop oder D-Latch. Es merkt sich auf Befehl eines Taktsignals den Zustand des Einganges und speichert ihn an seinem Ausgang. Der Ausgangszustand bleibt unverändert bzw. verriegelt, selbst wenn sich der Eingangszustand ändert, bis das nächste Taktsignal vorliegt. Der Eingang des D-Latches wird mit D, der Ausgang mit Q gekennzeichnet. Der Steuerbefehl wird in Form einer fallenden Taktflanke (Wechsel von 1 zu 0) oder einer steigenden Taktflanke (Wechsel von 0 zu 1) zum Steuereingang geführt. Man nennt solche Bausteine flankengetriggerte Bausteine (engl. edge-triggered devices). Beim D-Latch folgt der Ausgang dem Eingang, wann immer das Taktsignal 1 ist.

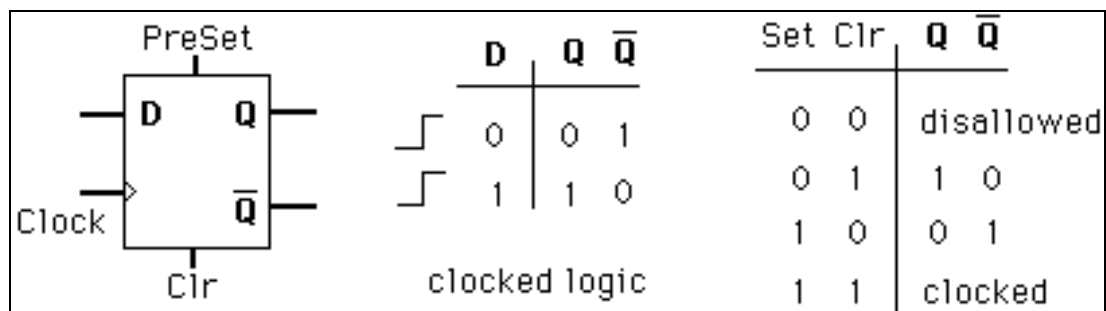


Abbildung 4-1. Symbol und Wahrheitstabelle eines D-Latches

Die Daten am Eingang D erscheinen beim Auftreten des Taktsignals an den Ausgängen Q und  $\bar{Q}$ . Die Wahrheitstabelle für ein D-Latch ist rechts des Symbols abgebildet. Einige D-Flip-Flops weisen noch einen Preset- und Clear-Eingang auf, wodurch der Ausgang unabhängig vom Taktsignal auf die Stellung 1 oder 0 gesetzt werden kann. Im Normalbetrieb werden diese zwei Eingänge auf 1 gesetzt, so dass es zu keiner Störung

mit der Steuerung des Flip-Flops über D- und Taktsignaleingang kommen kann. Mit dem Preset- bzw. Clear-Eingang kann jedoch das Flip-Flop unabhängig vom D- und Taktsignaleingang gelesen und zurückgesetzt werden.

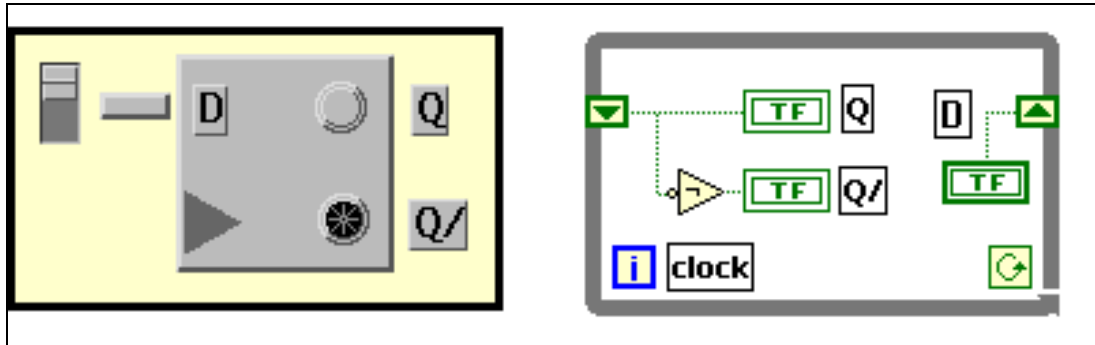


Abbildung 4-2. LabVIEW-Simulation eines D-Latches

In LabView können Sie das D-Latch durch ein Schieberegister innerhalb einer While-Schleife simulieren. Das rechte Terminal gilt als D-Eingang und das linke Terminal als Ausgang Q. Das Komplement  $\bar{Q}$  wird mit einem Inverter am Q-Ausgang gebildet. Der Taktsignaleingang entspricht dem Schleifenindex [i]. Durch eine Boolesche Konstante außerhalb der Schleife können Sie den Ausgang setzen oder zurückstellen. Beim oben abgebildeten **D Latch.vi** ist das Bedingungs-Terminal nicht angeschlossen, damit das Flip-Flop, wenn man es startet, nur einmal ausgeführt wird.

### Schieberegister (Shift Register)

In der digitalen Elektronik ist ein Schieberegister eine Kaskade von 1-Bit Speichern. Jeder 1-Bit Speicher kopiert dabei den Bitgehalt seines Nachbarn auf Befehl eines Taktsignals.

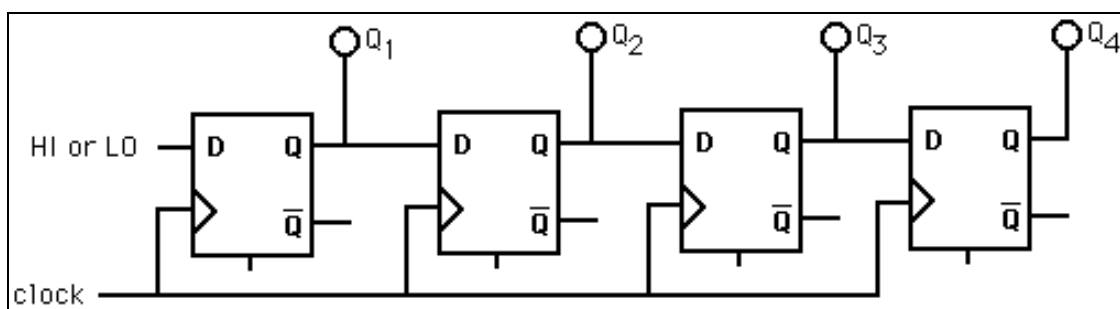


Abbildung 4-3. 4-Bit Schieberegister

Die Bit-Speicher an den jeweiligen Enden haben nur einen Nachbarn. Das Eingangssignal (0 oder 1) wird von einer externen Quelle bezogen und der Ausgang  $Q_4$  am Ende des Schieberegisters wird nach aussen geführt. Das folgende Beispiel zeigt ein 4-Bit Schieberegister mit Ausgangszustand [0000] und Eingang [1]:

Takt	Q1	Q2	Q3	Q4
$n$	0	0	0	0
$n + 1$	1	0	0	0
$n + 2$	1	1	0	0
$n + 3$	1	1	1	0
$n + 4$	1	1	1	1

Um in LabVIEW eine Kaskadenschaltung von D-Latches, wie oben gezeigt, zu realisieren, müssen dem Schieberegister zusätzliche Elemente hinzugefügt werden. In unserem Beispiel ist es ein 4-Bit Register. **Shift.vi** führt die oben aufgeführte Sequenz durch.

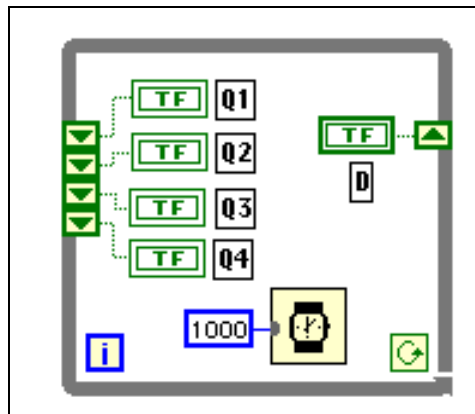


Abbildung 4-4. Blockdiagramm für ein 8-Bit Schieberegister

Auf einfache Weise können dem Schieberegister zusätzliche Elemente hinzugefügt werden, um so größere Registerbreiten zu simulieren. Das folgende VI, **Bucket.vi**, simuliert eine "Eimerketten-Schaltung", bei welcher ein einzelnes Bit zum Eingang D geführt wird, von wo es weiterwandert, bis es zum Ausgang  $Q_8$  kommt und dort ausgeworfen wird.

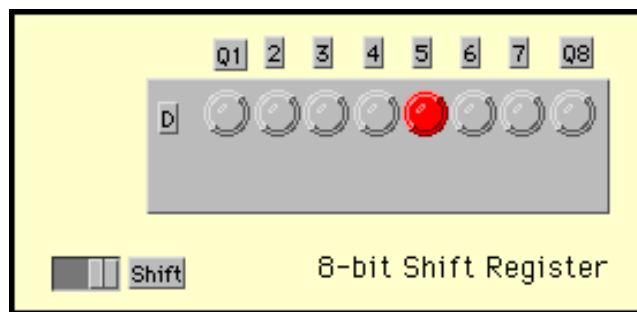


Abbildung 4-5. Front Panel einer 8-Bit-Schieberegister-Simulation

## LabVIEW-Knacknuss

Entwerfen Sie ein VI mit folgender Funktion: Nachdem "der Eimer" das letzte Bit durchlaufen hat, soll ein neuer "Eimer" zum Eingang D geführt werden und der Prozess soll wieder von neuem beginnen.

## Ringzähler

Wird der Ausgang eines Schieberegisters zum Eingang zurückgeführt, so wird nach  $n$  Zyklen der Ausgang wiederholt und das Schieberegister ist zu einem Zähler geworden. Der Name Ringzähler kommt von der Rückführung des Ausganges zum Eingang. Ein 4-Bit Ringzähler führt den letzten Ausgang  $Q_4$  direkt zum Eingang D des Schieberegisters zurück.

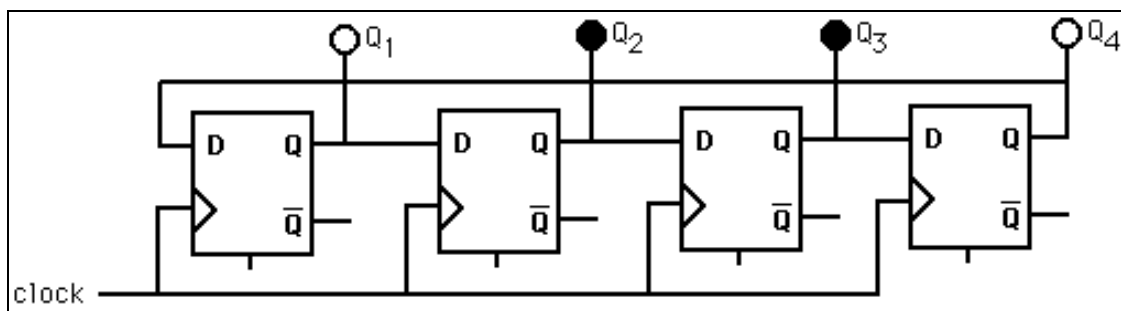
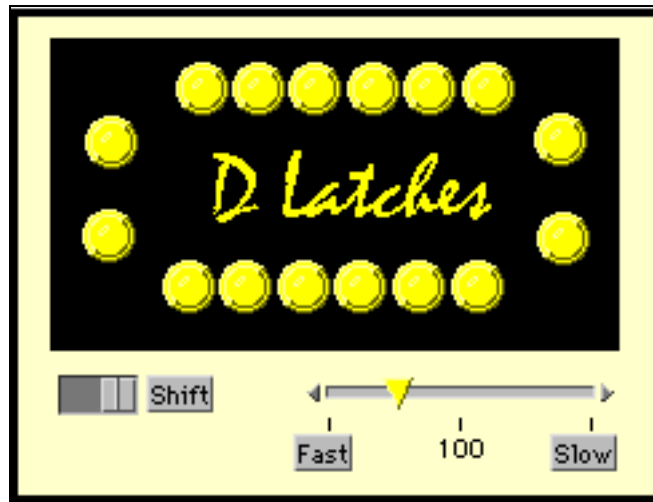


Abbildung 4-6. 4-Bit Ringzähler aufgebaut mit integrierten Schaltungen

In Abbildung 4-6 sind die Ausgänge auf 0110 eingestellt worden. Starten Sie **Rotate.vi** und beobachten Sie, wie die Ausgänge von 0110 auf 0011, auf 1001, auf 1100 und zurück auf 0110 wechseln. Es werden vier Schleifen durchlaufen, weshalb der Zähler Modulo-4-Ringzähler genannt wird. Im Falle, dass diese vier Ausgänge zu der Ansteuerung eines Schrittmotors geführt werden, ergibt jede Änderung des Ausganges ein Vorrücken des Motors um einen Schritt. Ein Schrittmotor mit 400 Schritten würde dann bei jedem Taktsignal um 0.9 Grad rotieren. Ein gegenüber dem Ringzähler leicht veränderter Zähler ist der "switched tail ring counter". Bei diesem wird der letzte Komplementausgang  $\bar{Q}$  zum Eingang zurückgeführt. Modifizieren Sie **Rotate.vi** zu einem "switched tail ring counter" und speichern Sie ihn unter **Switch Tail Ring Counter.vi**.

Wie sieht der Zählzyklus eines "switched tail ring counter" aus?

Ringzähler werden häufig dann eingesetzt, wenn Ereignisse gleichmässig wiederholt werden müssen. Öffnen und starten Sie **Billboard.vi**, welches durch aufleuchtende Lichter ein Boot simuliert.



Sie können durch den Schieberegler die Geschwindigkeit der aufleuchtenden Lichter einstellen. Mit den 16 Booleschen Konstanten im Blockdiagramm legen Sie das Verfolgungsmuster fest.

**Bibliothek der VIs der Übung 4 (in der gleichen Reihenfolge wie bearbeitet aufgelistet)**

**D Latch.vi** (LabVIEW-Simulation eines D-Latches)

**Shift.vi** (4-Bit Schieberegister)

**Bucket.vi** (Simulation eines 8-Bit Schieberegisters)

**Rotate.vi** (4-Bit Ringzähler)

**Billboard.vi** (16-Bit Ringzähler)

## **Notizen**

## Übung 5 - Pseudo-Zufallszahlen-Generator

In der letzten Übung wurden einfache Ringzähler zur Bildung von Modulo-n-Zählern eingeführt. Bei den Schieberegistern, die wir in dieser Übung betrachten, werden Rückkopplungen innerhalb des Registers vorgenommen. Wählt man die richtigen Rückkopplungen, so erreicht man die maximale Zählkapazität am Zählerausgang ( $=2^N-1$ ). Für einen 8-Bit Zähler bedeutet dies:  $2^8-1=255$  mit  $N=8$ . Diese Schaltungen, häufig Pseudo-Zufallszahlen-Generatoren (engl. Pseudo-Random Number Generators, PRNG) genannt, weisen einige interessante Eigenschaften auf. Die erzeugten Bitmuster scheinen über den kurzen Bereich zufällig zu sein. In Wirklichkeit aber wiederholen sich die Zufallszahlen nach  $2^N-1$  Zyklen. Ausserdem tritt jedes Muster nur einmal während einer Sequenz von  $2^N-1$  Zahlen auf.

Pseudo-Zufallszahlen-Generatoren haben breite Anwendungen in der Computersicherheit, in der Kryptographie, in der Prüfung von Audiosystemen, in der Bitfehlerprüfung und in der Sicherung der Datenübertragung.

### Ein 6-Bit Pseudo-Zufallszahlen-Generator

Im folgenden Schaltbild sind die Ausgänge des 5. und 6. D-Latches durch eine exklusives NOR verknüpft und zum Eingang des Schieberegisters zurückgeführt. Am Anfang seien alle Ausgänge Null.

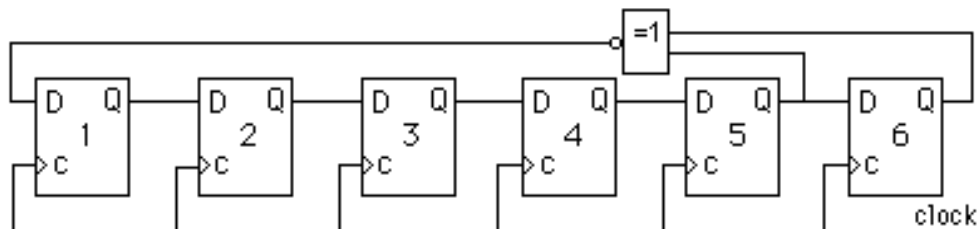


Abbildung 5-1. Ein 6-Bit-PRNG gebildet mit sechs D-Latches und einem NXOR-Gatter

Wenn  $Q_5$  und  $Q_6$  Null sind, so wird der Ausgang des NXOR 1 (siehe Übung 1). Dieses High gelangt zum Eingang D1 des Schieberegisters. Auf Befehl des Triggersignals verschieben sich alle Bits nach rechts und der Ausgangswert wechselt von 000000 auf 100000. Die Zustände an den Ausgängen  $Q_1...Q_6$  können bei den ersten paar Wiederholungen einfach nachvollzogen werden:

(000000)  
(100000)  
(110000)  
(111000)  
-----

Nach 63 Zyklen geht die Sequenz zum Ausgangszustand 000000 zurück.

Diese Schaltung kann einfach mit einem LabVIEW VI simuliert werden.

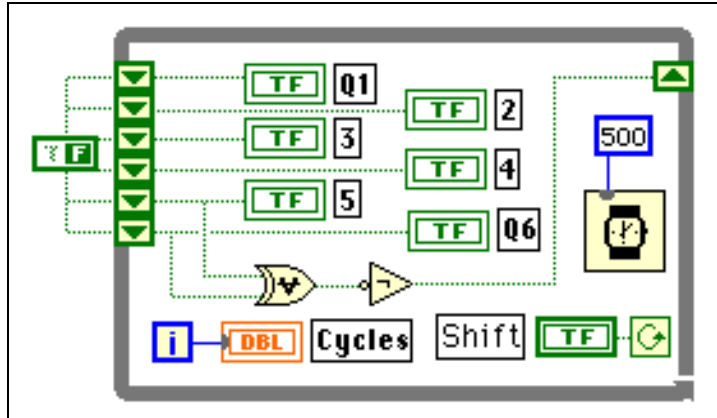


Abbildung 5-2. LabVIEW-VI zur Simulation eines 6-Bit PRNGs

Es wird dabei ein Schieberegister mit 6 Elementen in eine While-Schleife integriert. Ein OR-Gatter zusammen mit einem Inverter bilden das NXOR-Gatter, dessen Eingänge mit Q5 und Q6 verdrahtet werden. Der Schleifenindex verfolgt die Anzahl Wiederholungen und durch eine eingebaute Verzögerung von 500ms können die PRNG-Muster beobachtet werden. Bei der Durchführung des VIs, **6PRNG.vi**, werden Sie bemerken, dass Zyklus 0 und 63 dasselbe Bitmuster erzeugen; alle Bits sind nämlich Null.

### Ein 8-Bit Pseudo-Zufallszahlen-Sequenz

Bei einem 8-Bit PRNG werden die Ausgänge Q4, Q5, Q6 und Q8 miteinander NXOR-verknüpft, um so die maximale Zählkapazität von 255 zu erreichen.

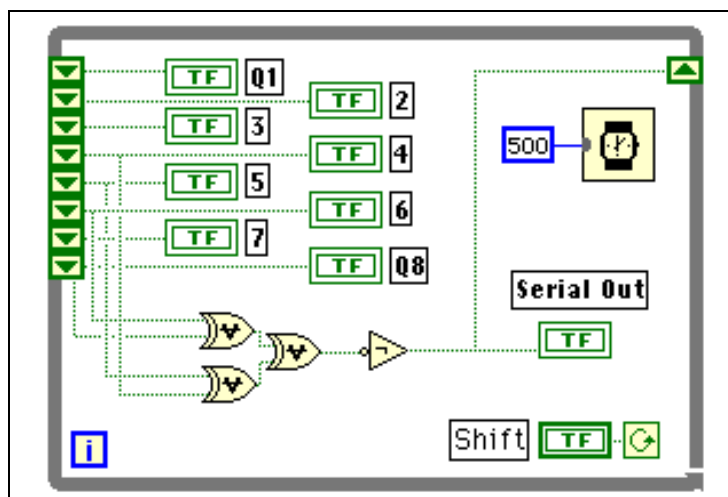


Abbildung 5-3. LabVIEW Simulation eines 8-Bit PRNG

Wie im vorhergegangenen Beispiel kann auch hier der parallele Ausgang mittels acht LED-Anzeigen verfolgt werden. Zusätzlich werden die Einsen und Nullen der Pseudo-Zufallsreihenfolge zu einem seriellen Ausgang geführt.

Viele Digitalschaltungen müssen mit allen möglichen Kombinationen von Einsen und Nullen geprüft werden. Eine "zufällige" Boolesche Reihenfolge von Einsen und Nullen liefert der serielle Ausgang. In dieser Konfiguration nennen wir die Schaltung Pseudo-Zufallsbit-Sequenz (engl. Pseudo-Random Bit Sequencer, PRBS). Im Front Panel des oben genannten VIs, **PRBS0.vi**, können Sie die Boolesche Bitanreihung (Serial Out) mittels einer LED-Anzeige verfolgen.

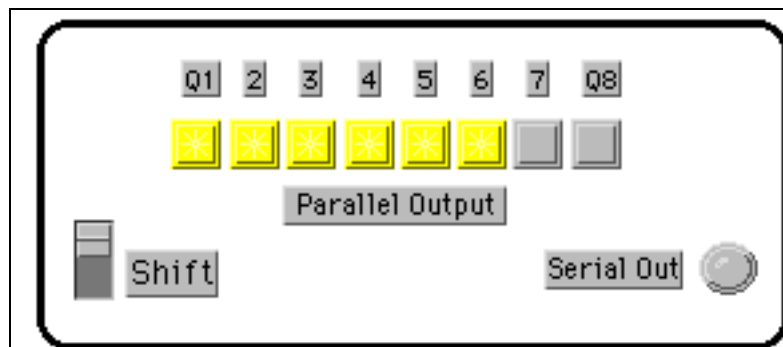


Abbildung 5-4. Front Panel des 8-Bit PRBS

Eine bessere Darstellung des Bitmusters kann mit einem Bit-Trace erreicht werden. Die Booleschen Bits werden in einen numerischen Wert von entweder 1 oder 0 umgewandelt und in einem LabVIEW-Diagramm angezeigt. Die untere Abbildung zeigt die Aufzeichnung der ersten 50 Bits von **PRBS.vi** mit einem Logiktrace.

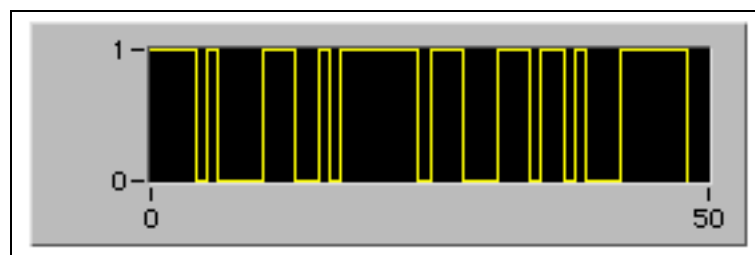
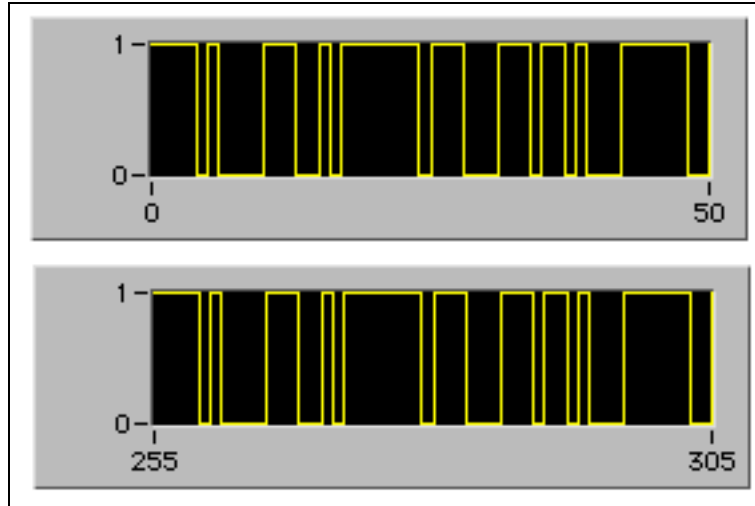


Abbildung 5-5. Serieller Ausgang des 8-Bit Pseudo-Zufallsbit-Sequenzers

Laser für die Datenübertragung werden mit PRBS-Kurvenformen geprüft. Es kann vorkommen, dass ein Laser bestimmte Sequenzen von Einsen und Nullen verriegelt oder dass Bits ausserhalb den Spezifikationen liegen. Der Laserausgang wird durch eine Fotodiode ermittelt, in ein digitales Signal umgewandelt und zu einem digitalen Komparator geführt. Gleichzeitig wird das PRBS-Signal zum anderen Eingang des Komparators geführt, wodurch Übertragungsfehler und Verriegelungen erkannt werden können.

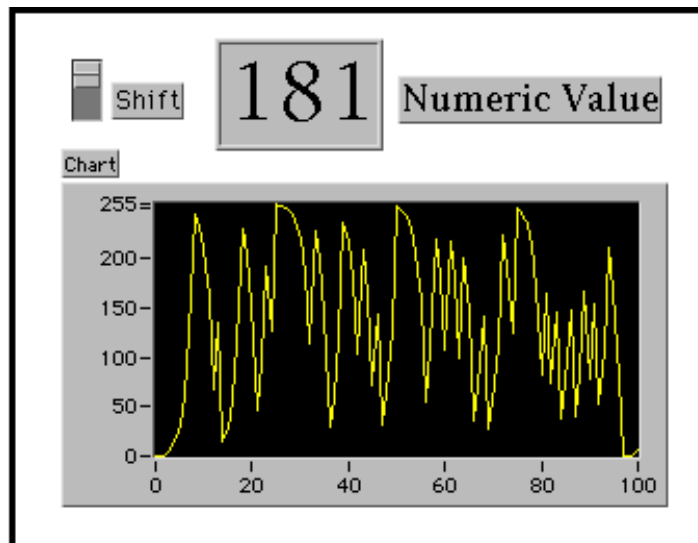
Es ist nun deutlich zu erkennen, dass sich das Bitmuster genau nach 255 Zyklen wiederholt. In **PRBS2.vi** zeigen zwei Diagramme das Bitmuster an. Indem Sie nun im zweiten Diagramm den Skalenbereich von 255 bis 305 festlegen, können Sie sich von dieser Eigenschaft überzeugen.



**Abbildung 5-6.** Vergleich der ersten 50 Bits eines PRBS mit den Bits von 255 – 305

### 8-Bit Pseudo-Zufallszahlen-Generator

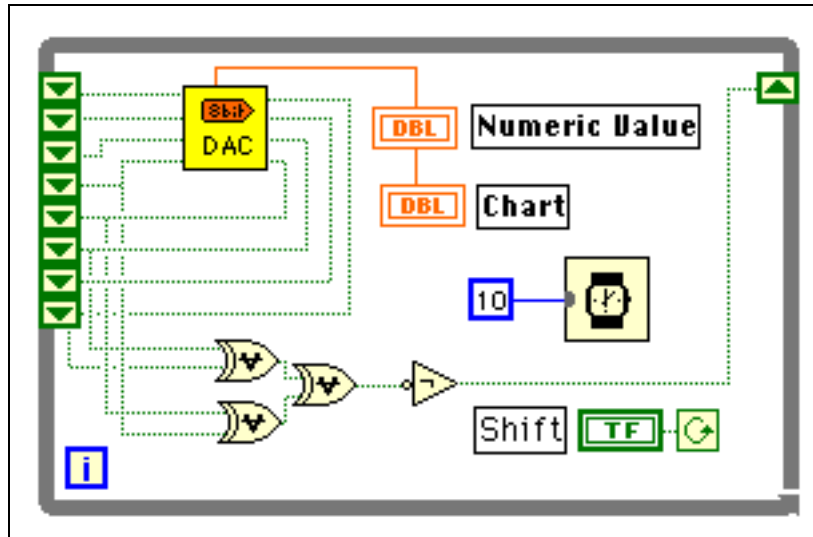
Durch einen Analog/Digital-Wandler kann die parallele Ausgabe der Pseudo-Zufallszahlen in eine numerische Zahl umgewandelt werden. Die parallel angeordneten Bits Q1...Q8 werden zu 1, 2, 4, 8, 16, 32, 64 und 128 konvertiert. Im folgenden VI werden die numerischen Werte auf einer dreistelligen Anzeige und einem Diagramm auf dem Front Panel angezeigt.



**Abbildung 5-7.** Numerische Ausgabe eines 8-Bit PRNG

Starten Sie **PRNG.vi** und beobachten Sie die PRNG-Reihenfolge der auftretenden Zahlen. Alle Zahlen von 0 bis 254 werden in der PRNG-Reihenfolge gefunden und nach genauerer Betrachtung erkennt man, dass jede Zahl nur einmal in der Reihenfolge auftritt. Erscheint die Reihenfolge in zufälliger Weise?

Das folgende Blockdiagramm zeigt die LabVIEW-Simulation eines 8-Bit PRNG. Beachten Sie, wie der DAC die numerischen Werte der Booleschen parallelen Ausgaben anzeigt.



**Abbildung 5-8.** LabVIEW Programm für ein 8-Bit PRNG mit Diagrammausgabe

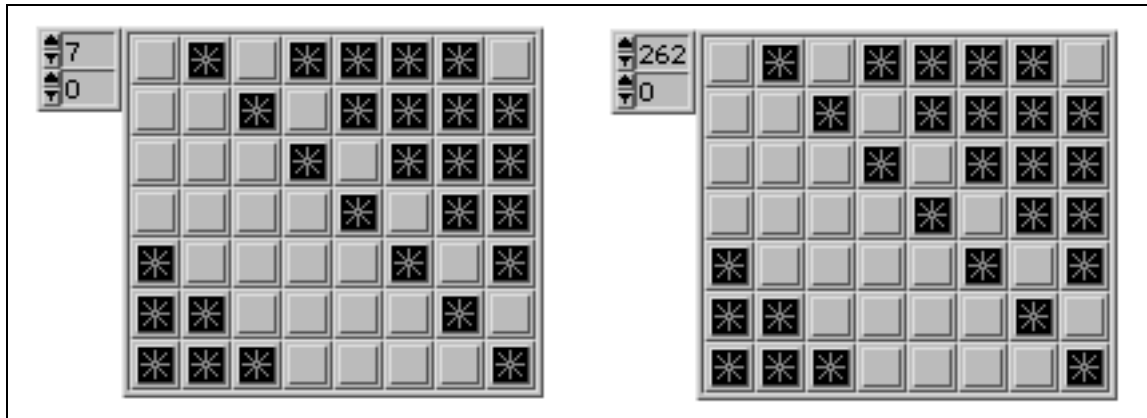
Das Diagramm zeigt die analoge Sequenz an. Über einen kurzen Bereich von 10-30 Zahlen ist die Ausgabe zufällig und tatsächlich auch mathematisch zufällig. Mit der Analogausgabe erscheint die Reihenfolge als weisses Rauschen. Der Vorteil von PRNG in der Audioprüfung ist, dass sich das Rauschen nach  $2^N-1$  Durchläufen wiederholt. Verstärker, wie z.B. digitale Gatter, können Kurzzeit-Speicher haben, jedoch keine Langzeit-Speicher. Die Analogausgabe eines PRNG wird als Test auf eine zu prüfende analoge Schaltung gegeben, deren Ausgabe mit den erwarteten Stufen der PRNG-Reihenfolge verglichen wird. Somit kann die Testschaltung auf mögliche Abweichungen (Fehler) geprüft werden.

### Codierung der digitalen Daten

Daten werden meist in Form von ASCII-Zeichen übertragen. Durch Hinzufügen eines Paritätsbits zu einem 7-Bit ASCII-Code ergibt sich eine 8-Bit Digitalzahl. Geldautomaten, elektronische Türverriegelungen und Computerpasswörter benutzen ASCII-Daten und irgendeine Verschlüsselung, um die Sicherheit zu wahren.

Der 8-Bit PRNG ist eine nützliche Schaltung für die Verschlüsselung von ASCII-Daten. Bis anhin haben wir als Ausgangswert für die PRNG-Reihenfolge jeweils den Default-Wert der LabVIEW Schieberegister genommen. Die Reihenfolge kann jedoch mit jedem beliebigen Start-Wert angefangen werden, ausser mit 11111111. Beträgt der Ausgangswert z.B. 01111010 oder 122 in numerischer, \$7A in hexadezimaler

Schreibweise oder "z" als ASCII-Zeichen, so wird die PRNG-Reihenfolge um diesen Wert versetzt, wiederholt sich aber in der üblichen Weise nach 255 Schleifen. Unten abgebildet ist ein Boolescher Array eines 8-Bit PRNGs, welcher bei Index 7 beginnt und die folgenden sechs Werte durchläuft. Beachten Sie, daß nach 255 Schleifen plus diesen Index, das ist  $7+255=262$ , die Reihenfolgen identisch und folglich voraussagbar sind.



**Abbildung 5-9.** Darstellung der Booleschen Arrays von 8-Bit Mustern der ersten acht Zahlen eines 8-Bit PRNGs und den Mustern der Schleifen von 262 bis 268

Angenommen, ein PIN oder ein Kennwort wird benutzt, um eine eindeutige numerische Kennziffer zu bilden. Der PRNG wird durch ein ASCII-Zeichen initialisiert und das PRNG wandelt dieses Eingangszeichen in ein verschlüsseltes Zeichen um, indem der PRNG N Schleifen durchläuft. Sind die N Schleifen ausgeführt worden, so enthalten die parallelen Ausgaben das verschlüsselte Zeichen. Wäre die PIN-Zahl 257, so würde gemäß obigen Beispiel das Zeichen "z" als "X" verschlüsselt werden. Es wird für jedes Zeichen einer Meldung ein neues Zeichen gebildet. Der Empfänger kennt den Verschlüsselungsalgorithmus und mit dem PIN kann die ursprüngliche Meldung dechiffriert werden.

**Bibliothek der VIs der Übung 5 (in der gleichen Reihenfolge wie bearbeitet aufgelistet)**

- 6PRNG.vi** (6-Bit PRNG)
- PRBS0.vi** (8-Bit Pseudo-Zufallszahlen – Sequenzer)
- PRBS.vi** (8-Bit PRBS mit seriellem Ausgang im Diagramm)
- PRNG.vi** (8-Bit PRNG mit Diagrammausgabe)
- PRNG7.vi** (8-Bit PRNG mit Array-Ausgabe)
- DAC8.vi** (8-Bit DAC SubVI)

## **Notizen**