

Digitale Schaltungen Aufarbeitung

Kajetan Weiß

27. November 2013

Inhaltsverzeichnis

I. Schaltnetze	4
1. Einführung	5
1.1. Binäre Entscheidungsdiagramme	5
1.2. Quine-McCluskey-Verfahren	7
2. Schaltnetzrealisierung	10
2.1. NAND und NOR Formen	10
2.2. Positive und Negative Logik	13
2.3. Multiplexer	13
2.4. ROM, PROM-Realisierung einer Schaltfunktion	16
2.5. PLA-Realisierungen	17
3. Laufzeitverhalten und Hazardanalyse	21
3.1. Totzeitmodell	21
3.2. Hazardklassifizierung	23
3.3. Behebung von strukturellen Hazardfehlern	25
3.4. Hazardfehler im Zeitdiagramm	25
3.5. Hazardanalyse mit Restfunktion	28
II. Schaltwerke	31
4. FlipFlops	32
4.1. Synchrones D-FlipFlop	32
4.2. Synchrones JK-FlipFlop	33
5. Endliche Automaten zur Schaltwerksmodellierung	34
5.1. Mealy Automat	34
5.2. Moore Automat	36
5.3. Medwedew Automat	37
5.4. Zustandsreduzierung	37
6. Schaltwerksynthese	40
6.1. Zustandskodierung	40
6.2. Ansteuerfunktionen	40

Inhaltsverzeichnis

7. Ausgewählte Schaltwerke	44
7.1. Register	44
7.1.1. Parallelregister	44
7.1.2. Schieberegister	45
7.2. Zählerschaltungen	47
7.2.1. Synchrone Zähler	47
7.2.2. Asynchrone Zähler	49
7.3. Speicher	50

Teil I.

Schaltnetze

1. Einführung

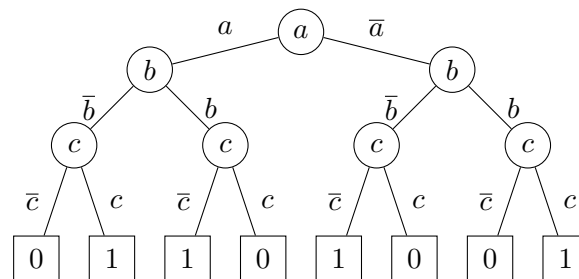
Ein Schaltnetz ist eine technische Realisierung einer mathematischen Zuordnung, Abbildung oder Funktion über einem binären Grundraum. Schaltnetze können mittels einer algebraischen booleschen Funktion, einer Schaltbelegungstabelle¹, Karnaugh-Veitch Diagrammen oder mittels binären Entscheidungsdiagrammen² dargestellt werden.

Um die Gatteranzahl bei der Realisierung möglichst gering zu halten, können Schaltnetze minimiert werden. Die Minimierung kann durch Umformung der algebraischen Darstellung, mit Hilfe der KV-Diagramme, Reduktion eines BDDs oder mittels des Quine-McCluskey-Verfahrens erfolgen. BDDs und das Quine-McCluskey-Verfahren werden in diesem Kapitel besprochen. Die Grundlagen für Schaltfunktionen und die Beschreibungen zu KV-Diagrammen und SBTs sind im Dokument Digitaltechnik Aufbereitung zu finden.

1.1. Binäre Entscheidungsdiagramme

Ein binäres Entscheidungsdiagramm ist die Darstellung einer logischen Funktion als Binärbaum. Einer Variablenbelegung entspricht dabei einem Pfad von der Wurzel zu den Blättern. Jedes Blatt repräsentiert einen Funktionswert zu einer bestimmten Variablenbelegung.

Beispiel:



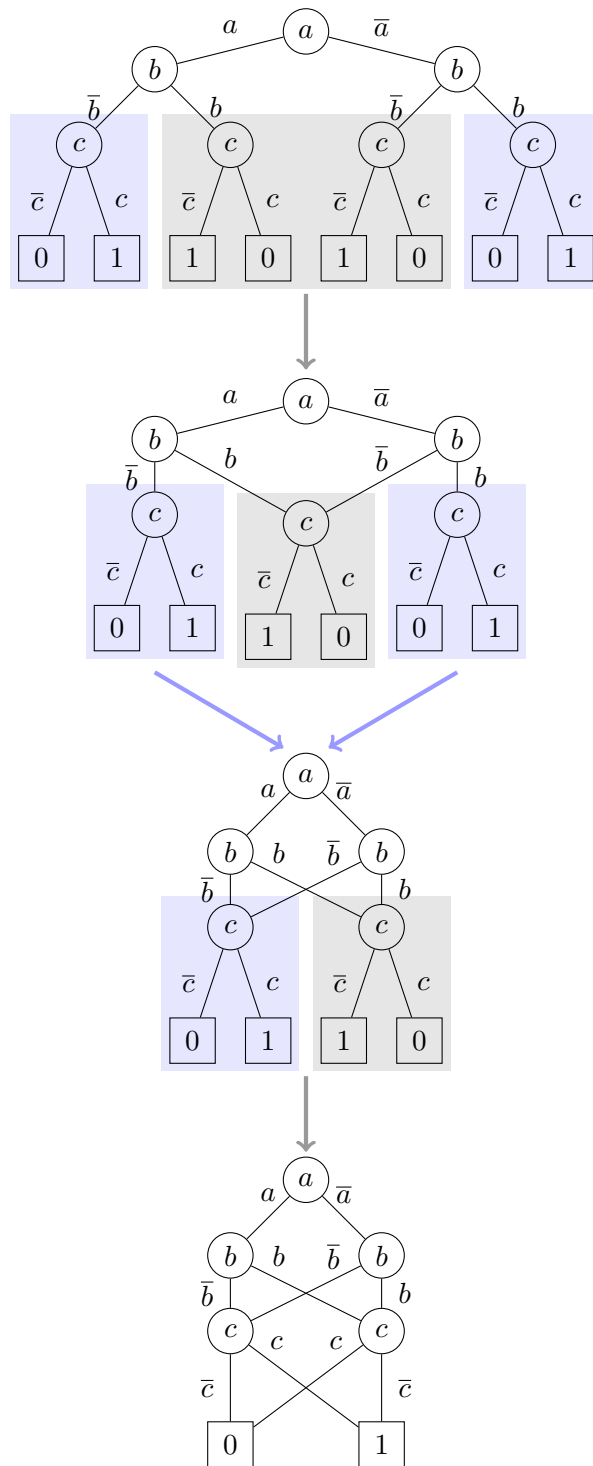
Der Entscheidungsbaum kann durch Verschmelzen gleicher Teilbäume und durch Löschen von irrelevanten Knoten reduziert werden. Zwei Teilbäume können miteinander verschmolzen werden, wenn sie gleich sind. Um die Übersicht zu behalten bietet es sich an möglichst große Teilbäume zu verschmelzen. Abbildung 1.1 zeigt dies anhand des oben angegebenen Beispiels.

¹'Schaltbelegungstabelle' kurz 'SBT'

²'Binäres Entscheidungsdiagramm' engl. 'binary decision diagram' kurz 'BDD'

1. Einführung

Abbildung 1.1.: Reduktion eines Entscheidungsdiagramms mittels Verschmelzung



1. Einführung

Führen die beiden ausgehenden Kanten eines Knotens auf denselben Knoten oder dasselbe Blatt so kann der Knoten gelöscht werden und dessen eingehende Kanten auf das Kind des gelöschten Knotens verzweigt werden.

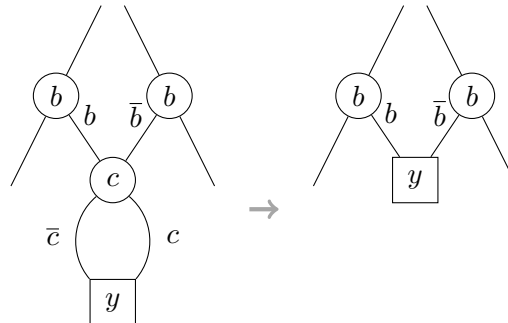


Abbildung 1.2.: Löschen eines Knotens

1.2. Quine-McCluskey-Verfahren

Nach dem Quine-McCluskey-Verfahren können systematisch Terme zusammengefasst werden, welche sich durch Negation einer einzigen Variablen unterscheiden. Das Quine-McCluskey-Verfahren eignet sich im Gegensatz zu KV-Diagrammen für Funktionen mit beliebig vielen Eingangsvariablen. Außerdem ist es leicht zu automatisieren. Beim händischen Minimieren von Funktionen sollten für Funktionen mit 4 oder weniger Eingangsvariablen KV-Diagramme verwendet werden, da diese übersichtlicher und für uns Menschen leichter zu lösen sind. Für Funktionen mit mehr als 4 Eingangsvariablen sollte das Quine-McCluskey-Verfahren eingesetzt werden.

Die Vereinfachung geschieht mittels Ordnungstabellen, welche schrittweise reduziert werden.

1. Die Minterme der Funktion werden aufgrund der Anzahl affirmierter³ Variablen zusammengefasst. So entstehen Gruppen von Termen, wobei alle Terme einer jeweiligen Gruppe gleich viele affirmierte Variablen besitzen. Die so entstandene Ordnung lässt sich in einer Ordnungstabelle notieren, wobei die Gruppen nach Anzahl der affirmierten Variablen aufsteigend sortiert werden.
2. Anschließend werden jeweils zwei Gruppen zusammengefasst, welche in der Sortierung direkt nebeneinander stehen. Eine Zusammenfassung zweier Terme wird vollzogen, wenn die Terme sich in ausschließlich einer Stelle durch Affirmation und Negation derselben Stelle unterscheiden. Die zusammengefassten Terme werden wiederum neuen Gruppen zugeordnet und in einer Ordnungstabelle nach gleichem

³Die Affirmation ist das Gegenteil der Negation. Eine affirmierte Variable ist also eine Variable als solche.

1. Einführung

Schema notiert. So entstehen Tabellen höherer Ordnung bis keine weitere Zusammenfassung möglich ist.

3. Terme, welche nicht weiter zusammengefasst werden konnten werden aufgrund dieser Eigenschaft Primimplikanten genannt. Für die minimierte Funktion sind allerdings nicht zwingend alle Primimplikanten notwendig. Primimplikanten, welche für die Funktion notwendig sind, werden Kernimplikanten genannt. Die Kernimplikanten werden anhand der Primimplikantentafel bestimmt. Dabei muss jeder Minterm von mindestens einem Kernimplikanten abgedeckt sein. Es gilt also eine minimale Anzahl an Kernimplikanten zu bestimmen, wobei jeder Minterm über die Menge der ausgewählten Kernimplikanten abgedeckt sein muss.

Beispiel: Die Funktion f mit den vier Eingangsvariablen a, b, c und d sei durch ihre Minterme $\{m_i \mid i \in \{0, 2, 5, 8, 10, 12, 13, 15\}\}$ eindeutig bestimmt.

$$f(a, b, c, d) = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}b\bar{c}d + \bar{a}b\bar{c}d + \bar{a}b\bar{c}d + \bar{a}b\bar{c}d + \bar{a}b\bar{c}d$$

Die Ordnungstabelle 0-ter Ordnung ist:

Term	Belegung				Gruppe	Notiz
	a	b	c	d		
0	0	0	0	0	0	
2	0	1	0	0	1	
8	0	0	0	1		
5	1	0	1	0	2	
10	0	1	0	1		
12	0	0	1	1		
13	1	0	1	1	3	
15	1	1	1	1	4	

Die Ordnungstabelle 1-ter Ordnung ist:

Term	Belegung				Gruppe	Notiz
	a	b	c	d		
0,2	0	—	0	0	0	
0,8	0	0	0	—		
2,10	0	1	0	—	1	
8,10	0	—	0	1		
8,12	0	0	—	1		prim
5,13	1	0	1	—	2	prim
12,13	—	0	1	1		prim
13,15	1	—	1	1	3	prim

Die Ordnungstabelle 2-ter Ordnung ist:

Term	Belegung				Gruppe	Notiz
	a	b	c	d		
0, 2, 8, 10	—	0	—	0	0	prim
0, 8, 2, 10	—	0	—	0		streichen

1. Einführung

die Primimplikantentafel ist also:

Primimplikant	Minterme								
	0	2	5	8	10	12	13	15	
8, 12				+ *		+			P_1
5, 13			+ #				+ *		K_1
12, 13						+	+ *		P_2
13, 15							+ #	+ #	K_2
0, 2, 8, 10	+ #	+ #		+ #	+ #				K_3

Die Minterme m_0 , m_5 , m_8 , m_{10} , m_{13} und m_{15} werden über die Terme K_1 , K_2 und K_3 abgedeckt. Wie an der Tafel zu erkennen ist, kann der Minterm m_{12} von P_1 oder P_2 abgedeckt werden. Daher muss einer der beiden Terme als Kernimplikant gewählt werden. Es gibt also zwei mögliche minimierte Formen der Funktion:

$$\begin{aligned}
 f(a, b, c, d) &= \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}b\bar{c}d + \bar{a}b\bar{c}d + \bar{a}b\bar{c}d + abcd \\
 &= \bar{a}\bar{b}d + \bar{a}\bar{b}c + acd + \bar{b}\bar{d} \\
 &= \bar{b}cd + \bar{a}\bar{b}c + acd + \bar{b}\bar{d}
 \end{aligned}$$

2. Schaltnetzrealisierung

Schaltnetze können systematisch über ein Binary Decision Diagram oder einer Schaltfunktion umgesetzt werden. NAND bzw. NOR Schaltkreise sind dabei physikalisch günstiger, da weniger Transistoren, Leistung und geringere Schaltzeiten benötigt werden. Näheres zu NAND/NOR Funktionen ist im Dokument Digitaltechnik Aufbereitung zu finden.

2.1. NAND und NOR Formen

Jede Schaltfunktion kann nach den Gesetzen von De Morgan mit NAND oder NOR Bausteinen umgesetzt werden. Dabei wird der Umstand genutzt, dass die doppelte Negation die ursprüngliche Aussage erhält. Im Folgenden ein Beispiel:

$$\begin{aligned}y &= a \cdot b + \bar{c} \cdot d \\&= \overline{\overline{a \cdot b + \bar{c} \cdot d}} \# \text{ doppelte Negation erhält die Aussage} \\&= \overline{\overline{a \cdot b} \cdot \overline{\bar{c} \cdot d}} \# \text{ Gesetz von De Morgan führt zu NAND} \\&= \overline{\overline{\overline{a \cdot b}} \cdot \overline{\overline{\bar{c} \cdot d}} \cdot \overline{\overline{\bar{c} \cdot d}}} \# \text{ Doppelte Negation für NOR Realisierung} \\&= \overline{\overline{\overline{a \cdot b}} + \overline{\overline{\bar{c} \cdot d}}} \\&= \overline{\overline{\overline{a \cdot b}} + \overline{\overline{\bar{c} \cdot d}}} \\&= \overline{(\overline{\overline{a \cdot b}}) + (\overline{\overline{\bar{c} \cdot d}})} \# \text{ NOR Realisierung} \\&= \overline{(\overline{\overline{a \cdot b}}) + (\overline{\overline{\bar{c} \cdot d}})}\end{aligned}$$

Ausgehend von einer Disjunktiven Form¹ kann also über die Doppelte Negation zu einer NAND Funktion umgebaut werden. Es gilt mit den Mintermenen m_i :

¹Näheres zu Disjunktiven Formen kann im Dokument Digitaltechnik Aufarbeitung nachgeschlagen werden.

2. Schaltnetzrealisierung

$$\begin{aligned} y &= m_1 + m_2 + \dots + m_n = \sum_{i=1}^n m_i \\ &= \overline{\overline{m_1 + m_2 + \dots + m_n}} = \overline{\sum_{i=1}^n \overline{m_i}} \\ &= \overline{\overline{m_1} \cdot \overline{m_2} \cdot (\dots) \cdot \overline{m_n}} = \prod_{i=1}^n \overline{\overline{m_i}} \end{aligned}$$

Als allgemeine Methode um von einer Disjunktiven Form zu einer NAND Form umzuformen kann angegeben werden:

1. erhalte Aussage mittels doppelter Negation
2. wende ein mal das Gesetz von De Morgan an
3. Ergebnis ist eine zweistufige NAND Form der Funktion

Ein Beispiel zur Überführung der Form einer Funktion in Disjunktiver Form zu einer NAND Form:

$$\begin{aligned} y &= ab + \overline{c}d + e \\ &= \overline{\overline{ab + \overline{c}d + e}} \\ &= \overline{\overline{ab} \cdot \overline{\overline{c}d} \cdot \overline{e}} \end{aligned}$$

Um die Umformung einer Disjunktiven Form auf eine NOR Form zu realisieren können folgende Regeln genutzt werden:

$$y = \sum_{i=1}^n m_i = \sum_{i=1}^n \overline{\overline{m_i}} = \sum_{i=1}^n \overline{\overline{\overline{m_i}}}$$

Als allgemeine Methode um von einer Disjunktiven Form zu einer NOR Form umzuformen kann angegeben werden:

1. erhalte Aussage mittels doppelter Negation der gesamten Funktion
2. erhalte die Aussage jedes Minterms mittels doppelter Negation des jeweiligen Terms
3. wende ein mal das Gesetz von De Morgan auf jeden doppelt negierten Minterm an
4. Ergebnis ist eine dreistufige NOR Form der Funktion

2. Schaltnetzrealisierung

Ein Beispiel zur Überführung einer Disjunktiver Form zu einer NOR Form:

$$\begin{aligned} y &= ab + \bar{c}d + e \\ &= \overline{\overline{ab + \bar{c}d + e}} \\ &= \overline{(\bar{a} + \bar{b}) + (c + \bar{d}) + e} \end{aligned}$$

Ausgehend von einer Konjunktiven Form mit den Maxtermen M_i kann eine NOR Form mit folgenden Regeln erstellt werden:

$$y = \prod_{i=1}^n M_i = \overline{\overline{\prod_{i=1}^n M_i}} = \overline{\sum_{i=1}^n \overline{M_i}}$$

Als allgemeine Methode um von einer Konjunktiven Form zu einer NOR Form umzuformen kann angegeben werden:

1. erhalte Aussage mittels doppelter Negation
2. wende ein mal das Gesetz von De Morgan an
3. Ergebnis ist eine zweistufige NOR Form der Funktion

Ein Beispiel zur Überführung der Form einer Funktion in Konjunktiver Form zu einer NOR Form:

$$\begin{aligned} y &= (a + b) \cdot (\bar{c} + d) \cdot e \\ &= \overline{\overline{(a + b) \cdot (\bar{c} + d) \cdot e}} \\ &= \overline{(a + b) + (\bar{c} + d) + \bar{e}} \end{aligned}$$

Ausgehend von einer Konjunktiven Form mit den Maxtermen M_i kann eine NAND Form mit folgenden Regeln erstellt werden:

$$y = \prod_{i=1}^n M_i = \overline{\overline{\prod_{i=1}^n M_i}} = \overline{\sum_{i=1}^n \overline{M_i}}$$

Als allgemeine Methode um von einer Konjunktiven Form zu einer NAND Form umzuformen kann angegeben werden:

1. erhalte Aussage mittels doppelter Negation

2. Schaltnetzrealisierung

2. erhalte die Aussage jedes Maxterms mittels doppelter Negation des jeweiligen Terms
3. wende ein mal das Gesetz von De Morgan auf jeden doppelt negierten Maxterm an
4. Ergebnis ist eine dreistufige NAND Form der Funktion

Ein Beispiel zur Überführung der Form einer Funktion in Konjunktiver Form zu einer NAND Form:

$$\begin{aligned}
 y &= (a + b) \cdot (\bar{c} + d) \cdot e \\
 &= \overline{\overline{(a + b)} \cdot \overline{(\bar{c} + d)} \cdot \bar{e}} \\
 &= \overline{(\bar{a} \cdot \bar{b}) \cdot (c \cdot \bar{d}) \cdot e}
 \end{aligned}$$

2.2. Positive und Negative Logik

Um die binären Werte $\{0, 1\}$ technisch abbilden zu können werden in der Halbleitertechnik die Signalpegel Low (L) und High (H) verwendet. Die Zuordnung von Signalpegel auf binären Wert ist dabei frei wählbar. Allerdings wirkt die Zuordnung sich auf die Logik aus. Tabelle 2.1 zeigt die Standard-Zuordnung der Signalpegel für die logische Grundfunktion. Je nachdem ob $L \mapsto 0$ oder $L \mapsto 1$ zugeordnet wird ergibt sich für die Standard-Zuordnung entweder das logische AND oder OR.

Signalpegel			Schaltfunktion					
Standard Zuordnung			positive Logik: AND			negative Logik: OR		
$\{L, H\} \mapsto \{L, H\}$			$L \mapsto 0$			$L \mapsto 1$		
b	a	y	b	a	y	b	a	y
L	L	L	0	0	0	1	1	1
L	H	L	0	1	0	1	0	1
H	L	L	1	0	0	0	1	1
H	H	H	1	1	1	0	0	0

Tabelle 2.1.: Positive und Negative Logik

Aufgrund dieser Zusammenhänge wird die positive Logik auch *wired AND* und die negative Logik auch *wired OR* genannt.

2.3. Multiplexer

Mit Multiplexern kann von mehreren Eingängen einer gezielt per Addressierung ausgewählt werden. Diese Funktion kommt beispielsweise in Bus-Systemen zum Einsatz. Je

2. Schaltnetzrealisierung

nachdem wieviele Werte adressiert werden sollen, müssen entsprechend viele Variablen als Adressvariablen gewählt werden. Jede Adressvariable steht dabei für eine Stelle einer Adresse.

Ein 2 zu 1 Multiplexer hat demnach zwei Eingänge, die es zu adressieren gilt. Demnach muss eine zusätzliche Eingangsvariable als Adressvariable zur Darstellung der Adressen 0 und 1 ergänzt werden. Abbildung 2.1 zeigt die Realisierung eines 2 zu 1 Multiplexers. Hat a den Wert 0 so wird der Wert x_0 an y übertragen. Hat a den Wert 1 wird der Wert von x_1 ausgewählt und an y übertragen.

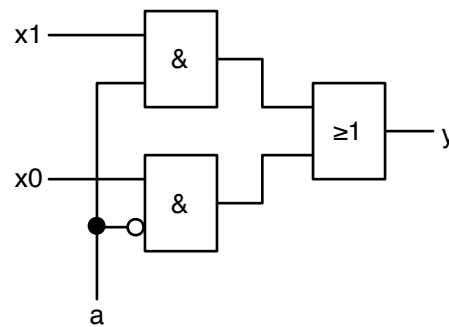


Abbildung 2.1.: Schaltbild eines 2 zu 1 Multiplexers

Um die Beschreibung von Multiplexern zu vereinfachen kann die Notation aus Abbildung 2.2 verwendet werden.

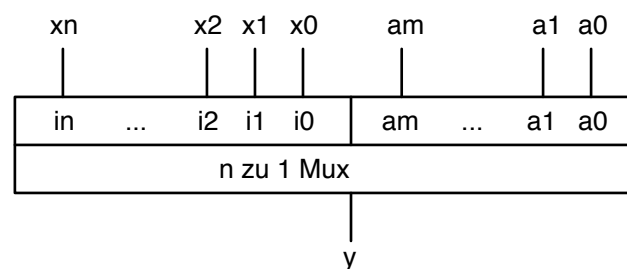


Abbildung 2.2.: Notation eines n zu 1 Multiplexers

Nach demselben Schema lassen sich n zu 1 Multiplexer aufbauen. Wobei n die Anzahl adressierbarer Elemente ist. Praktisch angewandt kann dies beispielsweise für eine Register zu Bus Verbindung genutzt werden. Wenn der Wert eines Registers einem Bus übergeben werden soll geschieht folgendes: In die Adressvariablen wird die Adresse des Registers geschrieben. Damit wird im Multiplexer das entsprechende Register ausgewählt. Die Ausgangsvariable erhält somit den Wert des Registers. Abbildung 2.3 zeigt das Schaltbild eines 4 zu 1 Multiplexers.

Aufgrund dieser zuordnenden Eigenschaft können Multiplexer auch zur Realisierung von Schaltfunktionen verwendet werden. Zu diesem Zweck werden den Adressen die Funktionswerte der zu realisierenden Funktion zugeordnet. Die Adressen entsprechen

2. Schaltnetzrealisierung

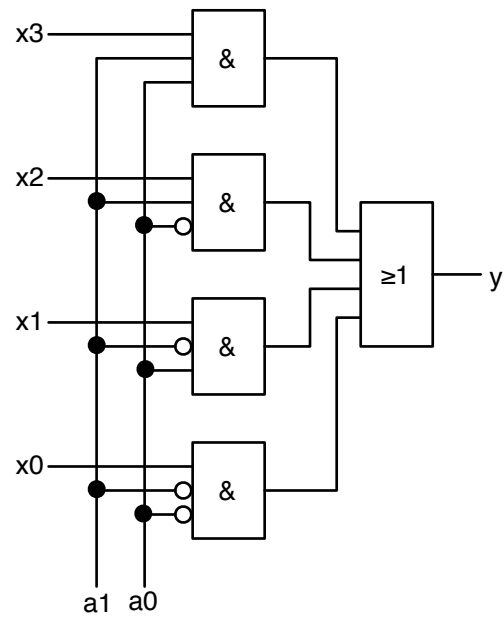


Abbildung 2.3.: Schaltbild eines 4 zu 1 Multiplexers

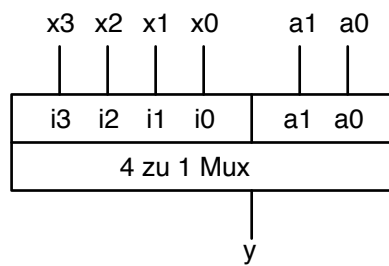


Abbildung 2.4.: Kurznotation des 4 zu 1 Multiplexers

2. Schaltnetzrealisierung

somit den Belegungen der Eingangsvariablen der Funktion in Disjunktiver Normalform. Da die Disjunktive Normalform aus der Schaltbelegungstabelle abgelesen werden kann, können die Funktionswerte aus der Schaltbelegungstabelle direkt auf den Multiplexer übertragen werden. Abbildung 2.5 zeigt ein Beispiel einer Funktion und deren Darstellung in einer Schaltbelegungstabelle, als Disjunktive Normalform und realisiert als Multiplexer.

i	0	1	2	3	4	5	6	7
c	0	0	0	0	1	1	1	1
b	0	0	1	1	0	0	1	1
a	0	1	0	1	0	1	0	1
y	1	0	1	0	1	0	0	1

$$y = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}bc + abc$$

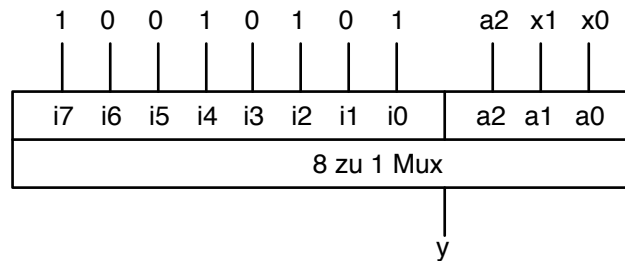


Abbildung 2.5.: Beispiel einer Funktion Realisiert als Multiplexer

Bei der Realisierung von Funktionen als Multiplexer können die Addressvariablen zur Minimierung des Multiplexers dienen. Dazu werden die Werte des Multiplexers abhängig von den Eingangsvariablen gesetzt. Mit diesem Verfahren kann die Funktion aus 2.5 mit dem Multiplexer aus Abbildung 2.6 realisiert werden.

2.4. ROM, PROM-Realisierung einer Schaltfunktion

ROM steht für Read Only Memory. PROM steht entsprechend für Programmable Read Only Memory. Es handelt sich um universell einsetzbare adressierende Einheiten zur Realisierung von Schaltnetzen. Jeder Adresse wird hierbei ein Speicherwort als Festwert bzw. Konstante zugeordnet. So können allen Variablenbelegungen einer Funktion der entsprechende Wert als Speicherwort im ROM zugeordnet werden. Dies ist Verleichbar mit dem nicht minimierten Verfahren zur Realisierung einer Funktion mit einem Multiplexer.

Beispiel: Es sei folgende Funktion gegeben:

$$f(x_0, x_1, x_2, x_3) = y = \prod \{M_0, M_2, M_3, M_4, M_5, M_8, M_{12}, M_{13}\}$$

Die Funktion soll mit einem PROM realisiert werden. Es werden also vier Adresseingänge benötigt, womit 2^4 Werte adressiert werden können. An den jeweiligen Adressen, die

2. Schaltnetzrealisierung

i	0	1	2	3	4	5	6	7		
c	0	0	0	0	1	1	1	1	b	0 0 1 1
b	0	0	1	1	0	0	1	1	$\mapsto a$	0 1 0 1
a	0	1	0	1	0	1	0	1	y	1 0 \bar{c} c
y	1	0	1	0	1	0	0	1		

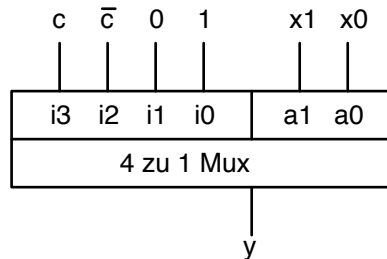


Abbildung 2.6.: Minimierter Multiplexer

den Indizes der Maxterme entsprechen, wird der Wert 0 einprogrammiert. An allen übrigen Adressen wird der Wert 1 einprogrammiert, da diese Adressen den Mintermen der Funktion entsprechen. Abbildung 2.7 zeigt das Schaltdiagramm für den PROM-Baustein.

2.5. PLA-Realisierungen

PLA steht für Programmable Logic Array. In einem PLA liegen alle Eingangsvariablen sowohl als solche als auch negiert vor. Die Variablenbelegungen können im Konjunktionsfeld des PLAS beliebig konjunktiv verknüpft werden. So entstehen rein konjunktiv verknüpfte Terme, welche als Minterme einer Funktion dienen können. Im Disjunktionsfeld des PLAS können die Minterme aus dem Konjunktionsfeld disjunktiv verknüpft werden. Die Verknüpfung der Variablen und Terme kann im Baustein programmiert werden. So kann jede Schaltfunktion in disjunktiver Form realisiert werden. Da jede Variable mit jeder anderen Variablen einen Minterm bilden kann, werden genauso viele Konjunktionsbausteine benötigt, wie Eingangsvariablen vorhanden sind. Es können beliebig viele Disjunktionsbausteine im Disjunktionsfeld gewählt werden um beliebig viele Funktionen mit den im Konjunktionsfeld erstellten Mintermen zu realisieren. Abbildung 2.8 zeigt das Schema eines PLA-Bausteins.

Beispiel: Die Funktionen f, g und h seien durch die Schaltbelegungstabelle 2.2 definiert. Die Funktion f soll mittels eines PLAS realisiert werden.

2. Schaltnetzrealisierung

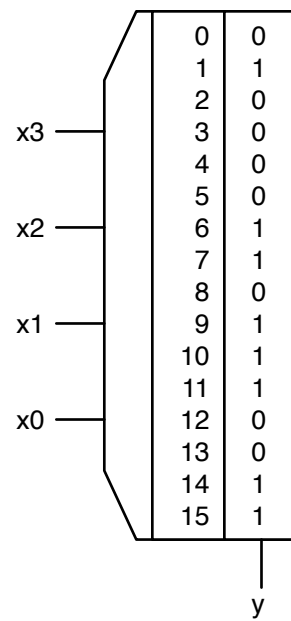
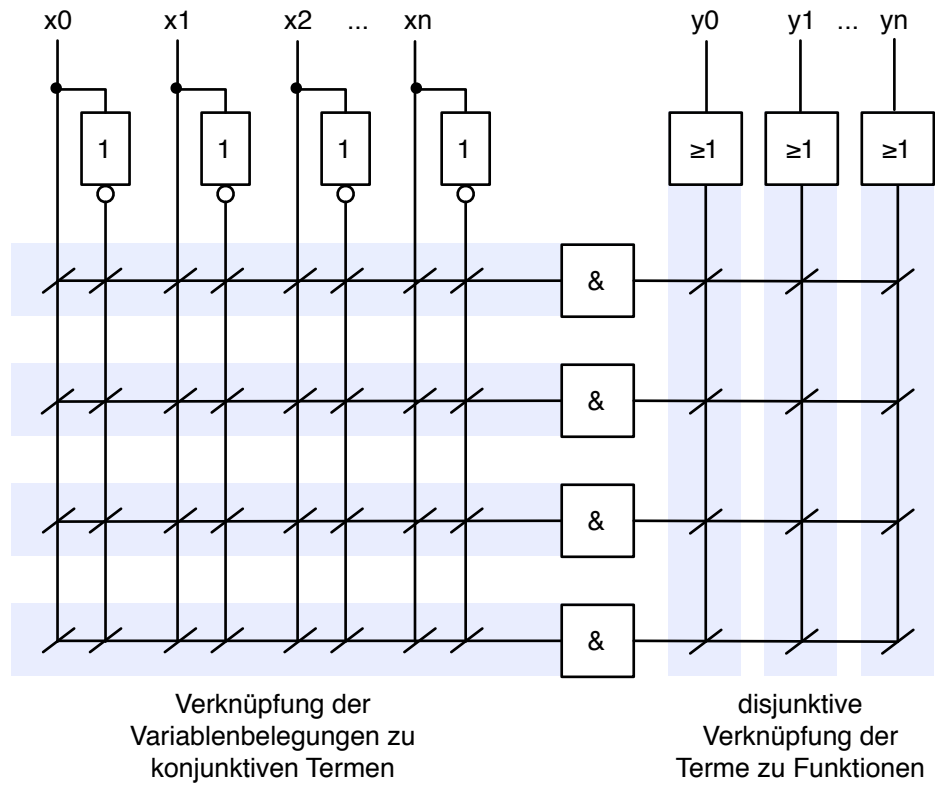


Abbildung 2.7.: PROM realisiert die angegebene Schaltfunktion

<i>i</i>	0	1	2	3	4	5	6	7
<i>c</i>	0	0	0	0	1	1	1	1
<i>b</i>	0	0	1	1	0	0	1	1
<i>a</i>	0	1	0	1	0	1	0	1
<i>f</i>	1	1	0	0	1	0	1	0
<i>g</i>	0	1	1	0	0	1	0	0
<i>h</i>	0	0	0	0	1	0	1	0

Tabelle 2.2.: Schaltbelegungstabelle der Funktionen f, g und h für das Beispiel zur Realisierung einer Funktion mittels eines PLA-Bausteins.

2. Schaltnetzrealisierung



✱ diese Stelle kann verbunden werden,
was zur Programmierung des PLAs führt.

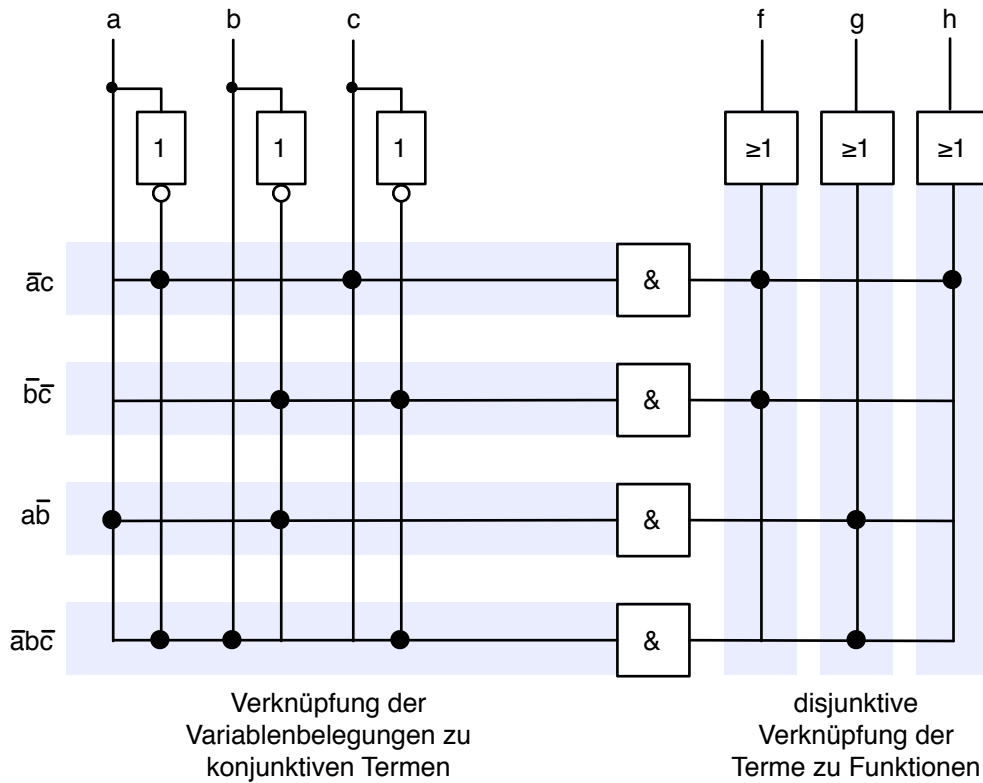
Abbildung 2.8.: Schema eines PLA-Bausteins

2. Schaltnetzrealisierung

Die Funktionen lassen sich wie folgt minimieren:

$$\begin{aligned} f &= \bar{a}c + \bar{b}\bar{c} \\ g &= a\bar{b} + \bar{a}b\bar{c} \\ h &= \bar{a}c \end{aligned}$$

Damit realisiert der PLA-Baustein aus Abbildung 2.9 die Funktionen f, g und h .




 diese Stelle ist verbunden und ist Teil des Programms des PLAs

Abbildung 2.9.: Beispiel eines programmierten PLA-Bausteins für die Funktionen f, g und h aus der Schaltbelegungstabelle 2.2

3. Laufzeitverhalten und Hazardanalyse

Jeder Zustand den eine Schaltfunktion einnehmen kann ist fest definiert. Aus rein logischer Sicht wird einer Variablenbelegung genau ein Ergebniszustand zugeordnet. Aus praktisch physikalischer Betrachtung braucht jedes Signal Zeit um von einer Schaltung verarbeitet zu werden. Vor allem ein Pegelwechsel von High zu Low und von Low zu High verbraucht Laufzeit. Die Signallaufzeit insgesamt ist abhängig von der Art der Gatter, die das Signal verarbeiten, die Anzahl der Stufen, die das Signal durchlaufen muss und abhängig von der Beschaffenheit der Leitungen in Art und Länge.

Je nach Eingangssignal können die Laufzeiten unterschiedlich sein. So können Fehlimpulse entstehen, die am Ausgang der Schaltfunktion kurzzeitig zu falschen Ergebnissen führen. Solche Fehler werden Hazard-Fehler genannt. Im folgenden wird die Hazardanalyse anhand der Beispielschaltfunktion aus Abbildung 3.1 erläutert.

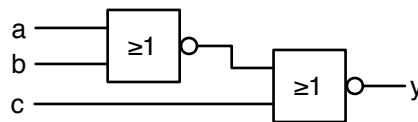


Abbildung 3.1.: Schaltdiagramm zur Hazardanalyse

Die Schaltfunktion zum Schaltdiagramm in Abbildung 3.1 lautet:

$$\begin{aligned} y &= \overline{\overline{a + b} + c} \\ &= (a + b) \cdot \bar{c} \end{aligned}$$

3.1. Totzeitmodell

In Abbildung 3.2 wird jedem Gatter und jeder Leitung eine Abstrakte Laufzeit τ_i zugeordnet. Aus diesen Laufzeiten lassen sich die Laufzeiten der Pfade von einem Eingangssignal zum Ausgang bestimmen. Jedem Pfad wird eine entsprechende Pfadvariable zugeordnet. Im Beispiel sind dies die Pfadvariablen a_1 , b_1 und c_1 , welche den jeweiligen Pfaden vom Signaleingang zum Ausgang zugeordnet werden.

3. Laufzeitverhalten und Hazardanalyse

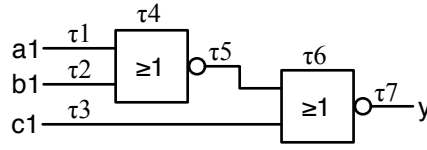


Abbildung 3.2.: Schalt diagramm zur Hazardanalyse mit eingetragenen Laufzeiten τ_i

So ergibt sich für die Laufzeiten der Pfade:

$$\tau_{a_1} = \tau_1 + \tau_4 + \tau_5 + \tau_6 + \tau_7$$

$$\tau_{b_1} = \tau_2 + \tau_4 + \tau_5 + \tau_6 + \tau_7$$

$$\tau_{c_1} = \tau_3 + \tau_6 + \tau_7$$

Jeder Pfad wirkt sich auf den Funktionswert aus. Da jeder Pfad eine unterschiedliche Laufzeit haben kann, können so Hazard-Fehler entstehen. Die Pfadfunktion¹ f wird aus der logischen Funktion y in Abhängigkeit der Pfade abgeleitet. Die Trennung zwischen Logikfunktion, Logikvariablen und Pfadfunktion, Pfadvariablen wird Totzeitmodell genannt.

Logikvariablen: $X = (a, b, c)$

Pfadvariablen: $P = (a_1, b_1, c_1)$

$$y(X) = (a + b) \cdot \bar{c}$$

$$f(P) = (a_1 + b_1) \cdot \bar{c_1}$$

Um den Übergang einer Variablenbelegung zu einer anderen auf mögliche Hazard-Fehler zu überprüfen werden die Logikfunktion und die Pfadfunktion getrennt betrachtet. Eine Betrachtung gilt von einem Anfangszustand zu einem Endzustand. Ein Zustand einer logischen Funktion ist mit der zustandsspezifischen Variablenbelegung definiert.

In der Beispielfunktion kann ein Hazard-Fehler beim Übergang der Variablenbelegungen $(a = 0, b = 1, c = 1)$ zu $(a = 1, b = 0, c = 0)$ entstehen. Um festzustellen unter welchen Bedingungen ein Hazardfehler auftritt wird die Pfadfunktion im Bezug auf diesen Übergang untersucht. P_a bezeichnet dabei den Anfangs- und P_e den Endzustand.

$$f(P) = (a_1 + b_1) \cdot \bar{c_1}$$

$$P = (a_1, b_1, c_1)$$

$$P_a = (0, 1, 1)$$

$$P_e = (1, 0, 0)$$

¹die Pfadfunktion wird auch transient switching function genannt

3. Laufzeitverhalten und Hazardanalyse

Eine mögliche Zustandsänderung ist eine Abfolge von Zwischenzuständen vom Anfangszustand zum Endzustand. Für die Folge gilt, dass von jedem Zustand zum nächsten sich die Variablenbelegung genau einer Variablen ändert. Außerdem gilt für die Folge insgesamt, dass jede Variable sich maximal ein mal ändert.

Um die möglichen Zustandsübergänge leichter zu erkennen wird das KV-Diagramm verwendet. Um die Bedingungen aus vorigem Absatz einzuhalten werden alle kürzesten Pfade von Anfangs zu Endzustand betrachtet. In Abbildung 3.3 ist der Anfangs- und Endzustand markiert. Die Pfeile zeigen die möglichen Zustandsübergänge. Alle Zustandsübergänge, die durch grüne Pfeile gekennzeichnet sind, enthalten keinen Hazard. Der Zustandsübergang, der durch die Abfolge roter Pfeile gekennzeichnet ist, enthält einen Hazard. Das Ausgangssignal würde bei diesem Zustandsübergang von 0 auf 1 auf 0 auf 1 wechseln.

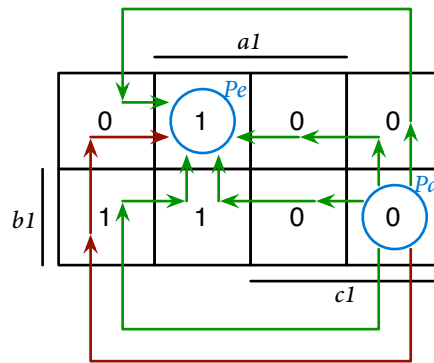


Abbildung 3.3.: KV-Diagramm zur Hazardanalyse der Pfadfunktion $f(P) = (a_1 + b_1) \cdot \bar{c}_1$

Anhand des Pfades im Diagramm aus 3.3 wird abgelesen unter welchem Umstand dieser Hazard-Fehler auftritt.

1. erster roter Pfeil \mapsto Wechsel von c auf \bar{c}
2. zweiter roter Pfeil \mapsto Wechsel von b auf \bar{b}
3. dritter roter Pfeil \mapsto Wechsel von \bar{a} auf a

Damit der rote Pfad eingeschlagen wird muss also gelten: $\tau_{c_1} < \tau_{b_1} < \tau_{a_1}$. Nach dem Schaltbild aus Abbildung 3.2 ist eine solche Abfolge von Zuständen durchaus möglich.

3.2. Hazardklassifizierung

Hazards werden anhand ihrer Eigenschaften klassifiziert. Je nach dem wie sich ein Hazard auf das Ausgangssignal auswirkt wird zwischen folgenden Hazards unterschieden:

- statischer 0-Hazard: $f(P_a) = 1$ und $f(P_e) = 1$; beim Übergang von P_a zu P_e ist kurzfristig $f = 0$

3. Laufzeitverhalten und Hazardanalyse

- statischer 1-Hazard: $f(P_a) = 0$ und $f(P_e) = 0$; beim Übergang von P_a zu P_e ist kurzfristig $f = 1$
- dynamischer 0-1-Hazard: $f(P_a) = 1$ und $f(P_e) = 0$; beim Übergang von P_a zu P_e ist kurzfristig $f = 0$ und anschließend $f = 1$
- dynamischer 1-0-Hazard: $f(P_a) = 0$ und $f(P_e) = 1$; beim Übergang von P_a zu P_e ist kurzfristig $f = 1$ und anschließend $f = 0$

In diesem Beispiel aus Abbildung 3.3 handelt es sich also um einen dynamischen 1-0-Hazard.

Zusätzlich wird zwischen strukturellen und funktionalen Hazard-Fehlern unterschieden. Grundsätzlich kann ein struktureller Hazardfehler² behoben werden. Ein Funktionshazard kann dagegen niemals behoben werden.

Um zu entscheiden ob es sich um einen Funktionshazard oder einen Strukturhazard handelt wird die Logikfunktion untersucht. Eine Hazardfreie Struktur kann immer angegeben werden, wenn im KV-Diagramm der Logikfunktion ein Pfad gefunden werden kann, welcher alle 1en überdeckt. Der Pfad kann mehrere Ausläufer haben.

Wenn ein solcher Pfad gefunden werden kann, handelt es sich um einen Strukturhazard. Existiert kein solcher Pfad, handelt es sich um einen Funktionshazard. Abbildung 3.4 zeigt einen solchen Pfad im KV-Diagramm für die Beispielfunktion. In diesem Fall handelt es sich also um einen strukturellen, dynamischen 1-0-Hazard.

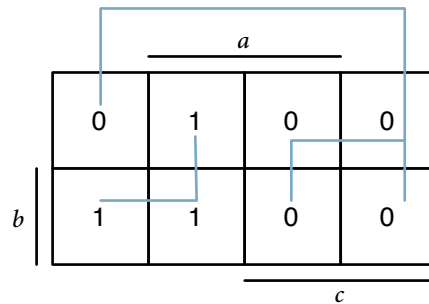


Abbildung 3.4.: KV-Diagramm zur Hazardanalyse der Logikfunktion $y(X) = (a + b) \cdot \bar{c}$

Formal kann außerdem der Übergangsbereich B und Übergangsterm u angegeben werden. Der Übergangsbereich bezeichnet alle möglichen Zustände, die die Funktion beim Übergang von einem Anfangs- in einen Endzustand einnehmen kann. Der Übergangsterm setzt sich aus der Konjunktion der Variablen zusammen, welche ihre Belegung während dem Übergang geändert haben. Für das Beispiel lassen sich Übergangsbereich

²siehe Abschnitt 3.3 zur Behebung von strukturellen Hazardfehlern

3. Laufzeitverhalten und Hazardanalyse

und Übergangsterm folgendermaßen aufschreiben:

$$\begin{aligned}X &= (a, b, c) \\X_a &= (0, 1, 1) \\X_e &= (1, 0, 0) \\B_X &= (-, -, -) \\u_X &= abc\end{aligned}$$

Für die Pfadfunktion gilt analog:

$$\begin{aligned}P &= (a_1, b_1, c_1) \\P_a &= (0, 1, 1) \\P_e &= (1, 0, 0) \\B_P &= (-, -, -) \\u_P &= a_1 b_1 c_1\end{aligned}$$

3.3. Behebung von strukturellen Hazardfehlern

Strukturelle Hazardfehler können behoben werden indem eine Schaltfunktion aufgebaut wird, welche für die betroffenen Pfadvariablen gleich viele Stufen enthält und die Primimplikanten der Logikfunktion nahtlos abdeckt. Die nahtlose Überdeckung bedeutet, dass der gefundene Pfad nahtlos im Funktionsterm abgebildet werden muss. In der Beispielfunktion muss die Funktion nach dem KV-Diagramm aus Abbildung 3.4 also auf folgende Funktion erweitert werden:

$$\begin{aligned}y(X) &= (a + b) \cdot \bar{c} \\&= (a + b) \cdot (\bar{a} + \bar{c}) \cdot (a + \bar{c}) \cdot (b + \bar{c}) \\&= \overline{(a + b) + (\bar{a} + \bar{c}) + (a + \bar{c}) + (b + \bar{c})}\end{aligned}$$

Damit kann eine äquivalente hazardfreie Schaltung wie in Abbildung 3.5 gezeigt realisiert werden.

Anmerkung: statische Strukturhazards können bei einer Disjunktiven Form auch mittels Disjunktion mit dem Übergangsterm behoben werden. Analog gilt für Konjunktive Formen die Konjunktion mit dem negierten Übergangsterm.

3.4. Hazardfehler im Zeitdiagramm

Abbildung 3.6 zeigt den 0-Hazardfehler im Zeitdiagramm. Kurz nachdem die Belegung der Variablen x geändert wird, sinkt kurzfristig der Wert der Funktion auf 0 und steigt anschließend wieder auf 1. 1 ist dabei der korrekte Wert der Funktion im Endzustand.

3. Laufzeitverhalten und Hazardanalyse

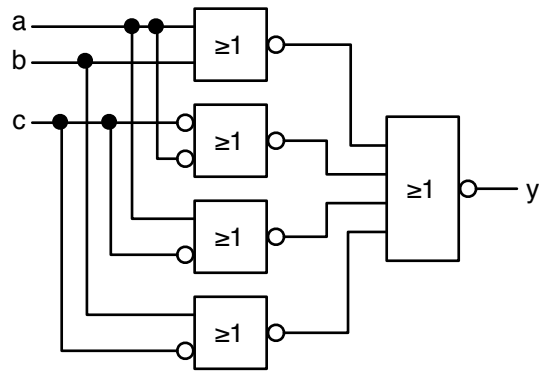


Abbildung 3.5.: Hazardfreies Schaltbild der Logikfunktion $y(X) = (a + b) \cdot \bar{c}$

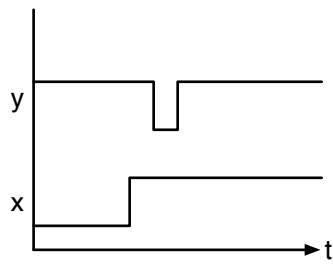


Abbildung 3.6.: 0-Hazardfehler im Zeitdiagramm

3. Laufzeitverhalten und Hazardanalyse

Abbildung 3.7 zeigt den 1-Hazardfehler im Zeitdiagramm. Kurz nachdem die Belegung der Variablen x geändert wird, steigt kurzfristig der Wert der Funktion auf 1 und fällt anschließend wieder auf 0. 0 ist dabei der korrekte Wert der Funktion im Endzustand.

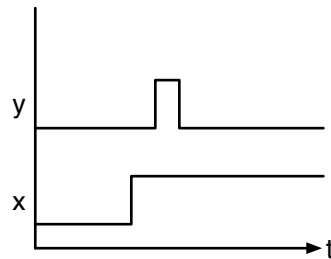


Abbildung 3.7.: 1-Hazardfehler im Zeitdiagramm

Abbildung 3.8 zeigt den 01-Hazardfehler im Zeitdiagramm. Kurz nachdem die Belegung der Variablen x geändert wird, sinkt kurzfristig der Wert der Funktion auf 0, steigt anschließend auf 1 und fällt wieder auf 0. 0 ist dabei der korrekte Wert der Funktion im Endzustand.

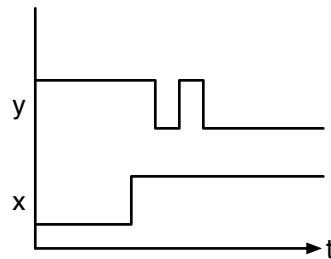


Abbildung 3.8.: 01-Hazardfehler im Zeitdiagramm

Abbildung 3.9 zeigt den 10-Hazardfehler im Zeitdiagramm. Kurz nachdem die Belegung der Variablen x geändert wird, steigt kurzfristig der Wert der Funktion auf 1, sinkt anschließend auf 0 und steigt wieder auf 1. 1 ist dabei der korrekte Wert der Funktion im Endzustand.

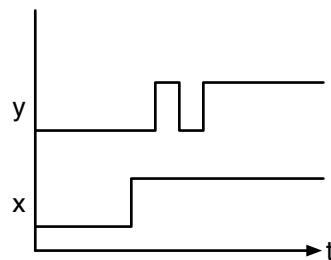


Abbildung 3.9.: 10-Hazardfehler im Zeitdiagramm

3.5. Hazardanalyse mit Restfunktion

In diesem Abschnitt wird anhand eines weiteren Beispiels besprochen, wie die Hazardanalyse auch nur mit dem betroffenen Teil der Funktion durchgeführt werden kann. Dies beschleunigt und vereinfacht den Analysevorgang.

Als Beispiel sei das Schaltdiagramm aus Abbildung 3.10 gegeben. Die Schaltung realisiert die Logikfunktion:

$$y = \bar{a} \bar{c} + bc$$

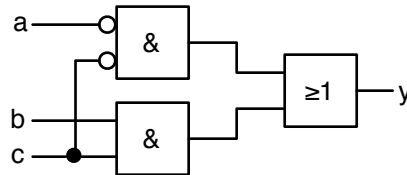


Abbildung 3.10.: Beispiel eines Schaltdiagramms zur Hazardanalyse

Ein Testlauf in einem Simulator ergab das folgende Laufzeitdiagramm aus Abbildung 3.11.

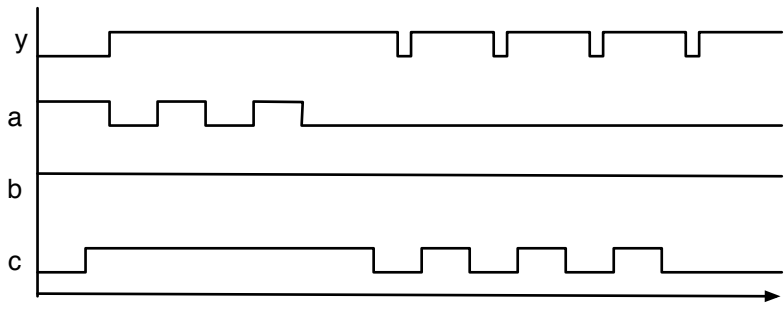


Abbildung 3.11.: Laufzeitdiagramm eines Testlaufs der Schaltung aus Abbildung 3.10

Merkwürdig sind die kurzen kurzfristigen 0-Stellen wenn die Belegung von c von 1 auf 0 gewechselt wird. Ob es sich hierbei um einen Hazardfehler handelt ist zu prüfen. Es wird zunächst der Ausgangszustand X_a und Endzustand X_e festgehalten. Damit lassen sich auch Übergangsbereich B_x und Übergangsterm u_x bestimmen.

$$X = (a, b, c)$$

$$X_a = (0, 1, 1)$$

$$X_e = (0, 1, 0)$$

$$B_x = (0, 1, -)$$

$$u_x = c$$

3. Laufzeitverhalten und Hazardanalyse

Es ändert sich bei der Logikfunktion nur eine Variable, wie am Übergangsbereich festzustellen ist. Wenn sich lediglich die Belegung einer Variablen ändert kann es keinen Hazard geben. Es handelt sich also nicht um einen Funktionshazard.

Der Vollständigkeit wegen wird hier die Restfunktion der Logikfunktion y_{Rest} für diesen Übergang betrachtet. Für die Restfunktion werden die Konstanten Belegungen aus dem Übergangsbereich für die entsprechenden Variablen eingesetzt. Daraus ergibt sich die Restfunktion einer Funktion. Auch hier ist zu erkennen, dass es sich um keinen Funktionshazard handeln kann, da die Restfunktion lediglich aus einer Konstanten besteht.

$$\begin{aligned} y_{Rest} &= y(a = 0, b = 1, c) \\ &= \bar{0} \cdot \bar{c} + 1 \cdot c \\ &= 1 \end{aligned}$$

Als nächstes muss die Pfadfunktion geprüft werden. Die Pfadfunktion ergibt sich aus der Logikfunktion wie folgt:

$$f(P) = \bar{a}_1 \bar{c}_1 + b_1 c_2$$

Anschließend werden Anfangs- und Endzustand, Übergangsbereich B_P und Übergangsterm u_P für die Pfadfunktion dokumentiert.

$$\begin{aligned} P &= (a_1, b_1, c_1, c_2) \\ X_a &= (0, 1, 1, 1) \\ X_e &= (0, 1, 0, 0) \\ B_x &= (0, 1, -, -) \\ u_x &= c_1 c_2 \end{aligned}$$

Damit lässt sich die Restfunktion f_{Rest} der Pfadfunktion aufstellen.

$$\begin{aligned} f_{Rest} &= f(a_1 = 0, b_1 = 1, c_1, c_2) \\ &= \bar{0} \cdot \bar{c}_1 + 1 \cdot c_2 \\ &= \bar{c}_1 + c_2 \end{aligned}$$

Um genau feststellen zu können unter welcher Bedingung der Hazardfehler entsteht wird das KV-Diagramm aus Abbildung 3.12 zur Restfunktion f_{Rest} untersucht. Im Diagramm wurden der Anfangs- und Endzustand markiert. Die Position der Zustände im Diagramm ergibt sich aus den Belegungen bei den jeweiligen Zuständen. Das Diagramm zeigt zwei mögliche Übergänge. Der mit roten Pfeilen markierte Übergang zeigt den statischen strukturellen 0-Hazardfehler. An diesem Übergang lässt sich auch die Bedingung für den Fehler ablesen. Der Hazardfehler tritt auf wenn:

$$\tau_{c_2} < \tau_{c_1}$$

3. Laufzeitverhalten und Hazardanalyse

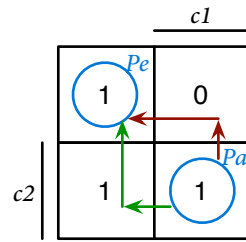


Abbildung 3.12.: KV-Diagramm Diagramm zur Restfunktion $f_{Rest} = \bar{c}_1 + c_2$

Da es sich um einen Strukturhazard handelt kann dieser behoben werden. Dazu gibt es die beiden angesprochenen Möglichkeiten: Die Primimplikanten nahtlos überdecken oder Disjunktion mit dem Übergangsterm. Folgende Funktionen sind hazardfrei:

y' über Disjunktion mit u_x (nicht äquivalent)

$$y' = \bar{a}\bar{c} + bc + c$$

y'' mit nahtloser Überdeckung der Primimplikanten

$$y'' = \bar{a}\bar{c} + bc + \bar{a}b$$

Funktionen, die alle Primimplikanten überdecken sind generell strukturhazardfrei.

Teil II.

Schaltwerke

4. FlipFlops

FlipFlops sind taktflankengesteuerte Speicherelemente. Zu einem bestimmten Zeitpunkt kann der Speicherwert des FlipFlops neu gesetzt oder der bisherige Wert erhalten werden. Der betreffende Zeitpunkt ist gekennzeichnet durch die positive oder negative Taktflanke (auch beide Taktflanken sind möglich). Es gibt noch weitere Speicherelemente. Darunter asynchrone Speicherelemente, die ohne Taktung arbeiten und Speicherelemente, welche während eines Taktzustandes beschaltet werden können. In diesem Dokument wird die Funktionsweise des synchronen JK-FlipFlops und des synchronen D-FlipFlops besprochen.

4.1. Synchrones D-FlipFlop

Abbildung 4.1 zeigt das Schaltsymbol und das Schaltverhalten des D-FlipFlops mit positiver Taktflanke. FlipFlops, welche bei negativer oder beiden Taktflanken angesteuert werden können, können analog betrachtet werden. Das Symbol \uparrow symbolisiert hier die positive Taktflanke. \downarrow symbolisiert entsprechend die negative Taktflanke.

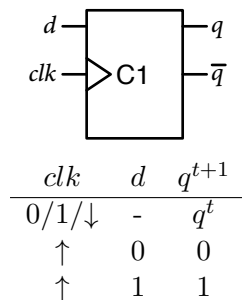


Abbildung 4.1.: Schaltungsdiagramm und Schaltdiagramm eines D-FlipFlops mit positiver Taktflanke

Das D-FlipFlop nimmt einen Wert am Eingang d entgegen und speichert diesen Wert bis er neu gesetzt wird. Der gespeicherte Wert kann am Ausgang q abgefragt werden. Zusätzlich gibt es den Ausgang \bar{q} , welcher die Negation des gespeicherten Wertes ausgibt.

Allgemeingültige Ansteuergleichungen für D-FlipFlops sind in folgender Tabelle erfasst.

4. FlipFlops

z^t	\mapsto	$z^t + 1$	D
0	\mapsto	0	0
0	\mapsto	1	1
1	\mapsto	0	0
1	\mapsto	1	1

4.2. Synchrones JK-FlipFlop

Das JK-FlipFlop bietet als Baustein eine weitaus größere Funktionalität als das D-FlipFlop. Abbildung 4.2 zeigt das Schaltsymbol und die Wahrheitstabelle zum JK-FlipFlop. J steht für „jump“ und K für „kill“.

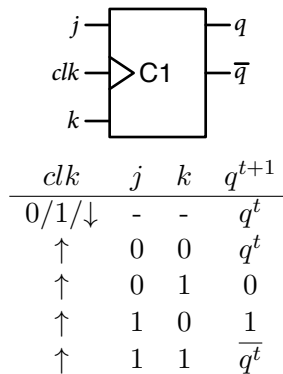


Abbildung 4.2.: Schaltungsdiagramm und Schaltdiagramm eines JK-FlipFlops mit positiver Taktflanke

Das JK-FlipFlop reagiert, wie alle FlipFlops, auf eine Ansteuerung zum Zeitpunkt einer Taktflanke. An der Wahrheitstabelle wird abgelesen welche Funktionalität das JK-FlipFlop bietet. Wird an jk 00 angelegt bleibt der bestehende Zustand erhalten. Wird 01 angelegt wird der aktuelle Zustand auf 0 gesetzt. Wird 10 angelegt wird der aktuell gespeicherte Wert auf 1 gesetzt. Bei der Belegung 11 ist das Resultat für den folgenden Zustand abhängig vom vorhergehenden Zustand. Es wird der negierte Zustand vom vorherigen Zustand gespeichert. War der gespeicherte Wert 1 wird 0 gespeichert. War der gespeicherte Wert 0 wird 1 gespeichert.

Allgemeingültige Ansteuergleichungen für JK-FlipFlops sind in folgender Tabelle erfasst.

z^t	\mapsto	$z^t + 1$	J	K	Eselsbrücke
0	\mapsto	0	0	*	no jump
0	\mapsto	1	1	*	jump
1	\mapsto	0	*	1	kill
1	\mapsto	1	*	0	no kill

5. Endliche Automaten zur Schaltwerksmodellierung

Präziser werden zur Schaltwerksmodellierung Automaten der Automatenklasse der Transduktoren verwendet. Transduktoren wandeln eine Eingabesequenz in eine Ausgabesequenz um. Sie haben also eine Übersetzerfunktion. Jeder Transduktor besteht aus Eingabealphabet X , Ausgabealphabet Y , Ausgabefunktion δ , Zustandsmenge Z , Zustandsübergangsfunktion λ , welche auch Überföhrungsfunktion genannt wird, und Startzustand z_a . Formal als 6-Tupel:

$$(X, Y, Z, \lambda, \delta, z_a)$$

Es gibt verschiedene Arten von Transduktoren. In den folgenden Abschnitten werden die Mealy-, Moore- und der Medwedew Transduktoren beschrieben. Die Automaten unterscheiden sich ausschließlic in der Art der Ausgabefunktion. Die Übergangsfunktion ist stets eine Abbildung von einem Ausgangszustand und einem Eingabezeichen auf einen Folgezustand. Formal gilt mit dem Ausgangszustand z^t zum Zeitpunkt t und Eingabezeichen x^{t+1} zum Zeitpunkt $t + 1$ folgt der Folgezustand z^{t+1} :

$$\delta : (z^t, x^{t+1}) \mapsto z^{t+1}$$

Schaltwerke können

- grafisch mittels Zustandsgrafen
- tabellarisch mittels Automatentabellen (kann aus Zustandsgraf abgelesen werden)
- algebraisch mittels Überföhrungsfunktion und Ausgabefunktion (kann aus Automatentabelle generiert werden)
- oder als Programm in einer Register-Transfer-Sprache

vollständig beschrieben werden.

5.1. Mealy Automat

Die dem Mealy-Automaten spezifische Ausgabefunktion ist zusammengesetzt aus aktuellem Zustand und folgendem Eingabesymbol. Formal gilt aus dem Ausgangszustand z^t zum Zeitpunkt t und Eingabezeichen x^{t+1} zum Zeitpunkt $t + 1$ folgt das Ausgabesymbol y^{t+1} :

$$\lambda_{mealy} : (z^t, x^{t+1}) \mapsto y^{t+1}$$

5. Endliche Automaten zur Schaltwerksmodellierung

Grafisch wird beim Mealy Automat an die Kanten von einem Zustand zum nächsten das entsprechende Eingabe- und entsprechende Ausgabesymbol notiert. Das Beispiel aus Abbildung 5.1 zeigt den Grafen eines Binärzahl-zu-Graycode-Transduktors. Das Symbol vor dem „/“ ist das Eingabesymbol und das Symbol nach dem „/“ ist das entsprechende Ausgabesymbol.

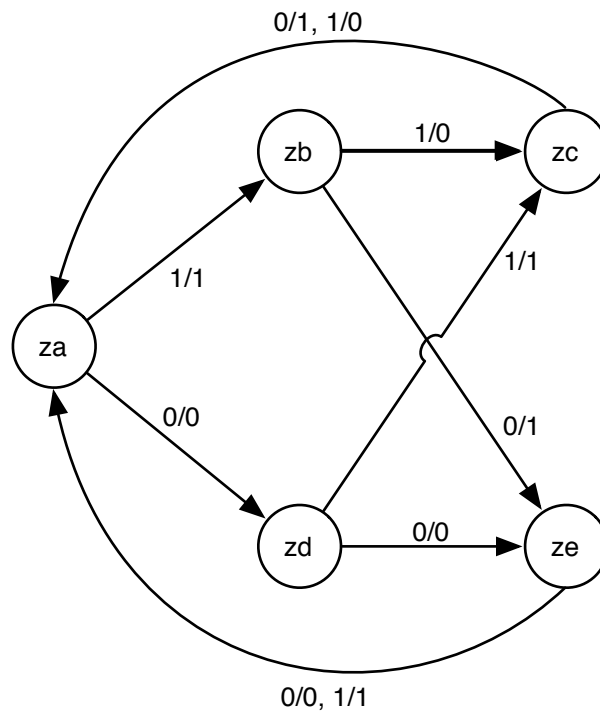


Abbildung 5.1.: Mealy Automat, welcher aus Binärzahlen Zahlen im Graycode erstellt

Tabellarisch kann der Mealy-Automat auf zwei Arten beschrieben werden.

- erste Variante: jede Zeile steht für einen Zustand; die Spalten entsprechen den Eingabesymbolen

z^t	$x \in \{0, 1\}$	
	0	1
z_0	$z_3/0$	$z_1/1$
z_1	$z_4/1$	$z_2/0$
z_2	$z_0/1$	$z_0/0$
z_3	$z_4/0$	$z_2/1$
z_4	$z_0/0$	$z_0/1$

- zweite Variante: jede Zeile steht für einen Zustand; die Spalten entsprechen den

Eingangs- und Ausgabevariablen und dem Folgezustand

z^t	x	z^{t+1}	y
z_0	0	z_3	0
z_0	1	z_1	1
z_1	0	z_4	1
z_1	1	z_2	0
z_2	0	z_0	1
z_2	1	z_0	0
z_3	0	z_4	0
z_3	1	z_2	1
z_4	0	z_0	0
z_4	1	z_0	1

5.2. Moore Automat

Die Ausgabefunktion des Moore Automats ist ausschließlich von dem aktuellen Zustand abhängig. Formal gilt aus einem Zustand z^t zum Zeitpunkt t folgt das Ausgabesymbol y^t :

$$\lambda_{moore} : z^t \mapsto y^t$$

Grafisch wird beim Moore Automat an die Kanten das entsprechende Eingabesymbol notiert. Die Bezeichnung der Zustände wird vor dem „ / “ in den Knoten, der dem entsprechenden Zustand entspricht geschrieben. Das Ausgabesymbol entsprechend jedes Zustandes wird nach dem „ / “ notiert. Das Beispiel aus Abbildung 5.2 zeigt einen Moore Automaten, welcher die Funktionsweise eines D-FlipFlops abbildet.

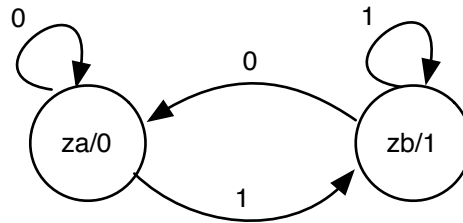


Abbildung 5.2.: Moore Automat, welcher ein D-FlipFlop repräsentiert

Die folgende Tabelle zeigt die tabellarische Darstellung des Automaten.

z^t/y	$x \in \{0, 1\}$	
	0	1
$z_0/0$	z_0	z_1
$z_1/1$	z_0	z_1

5.3. Medwedew Automat

Der Medwedew Automat kann als Vereinfachung des Moore-Automats betrachtet werden. Dabei sind die Ausgabesymbole gleich den Bezeichnern der Zustände. Formal gilt also

$$\lambda_{medwedew} : z^t \text{ mit } z^t = y^t$$

Abbildung 5.3 zeigt zur Demonstration denselben Automaten aus Abbildung 5.2 in Medwedew Darstellung.

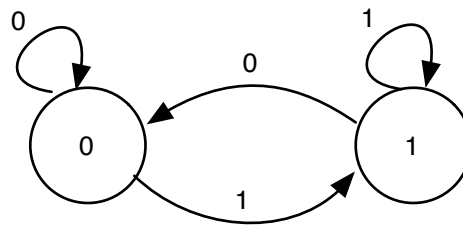


Abbildung 5.3.: Medwedew-Automat, welcher ein D-FlipFlop repräsentiert

Wegen dieser Vereinfachung ist in der realisierenden Schaltung eine Funktion weniger nötig. Ansonsten müsste nach Moore die Abbildung von Zuständen auf Ausgabewerte ebenfalls als Funktionsschaltung in das Realisierende Schaltwerk aufgenommen werden.

5.4. Zustandsreduzierung

Ziel ist es systematisch äquivalente Schaltwerke herzuleiten um sie auf bestimmte Aspekte hin zu optimieren. Äquivalente Schaltwerke zeichnen sich dadurch aus, dass bei allen Eingabesequenzen alle zugehörigen Ausgabesequenzen gleich sind. Automaten können minimiert werden, indem stabile kompatible Zustände miteinander verschmolzen werden. Kompatible Zustände sind Zustände, die das gleiche Folgeverhalten aufweisen. Für zwei Kompatible Zustände gilt, dass sie bei gleicher Eingabe in denselben Folgezustand übergehen. Stabile Zustände werden an Schleifen erkannt. Führt ein Zustand mit einer bestimmten Eingabe x in denselben Zustand wird er als stabil mit der Eingabe x bezeichnet.

Die Verschmelzung von Zuständen kann von einem Moore-Automaten zu einem Mealy-Automaten führen. In Abbildung 5.4 ist das Diagramm des Beispielautomaten angegeben, welcher im folgenden Abschnitt systematisch reduziert wird.

Die Tabelle 5.4 beschreibt den Automaten aus Abbildung 5.4. Stabile Zustände sind mit \square markiert. Nicht benutzte Zustands/Eingabesymbol-Kombinationen sind mit * gekennzeichnet. Diese sind somit mit allen anderen Kombinationen kompatibel, da sie nicht verwendet werden.

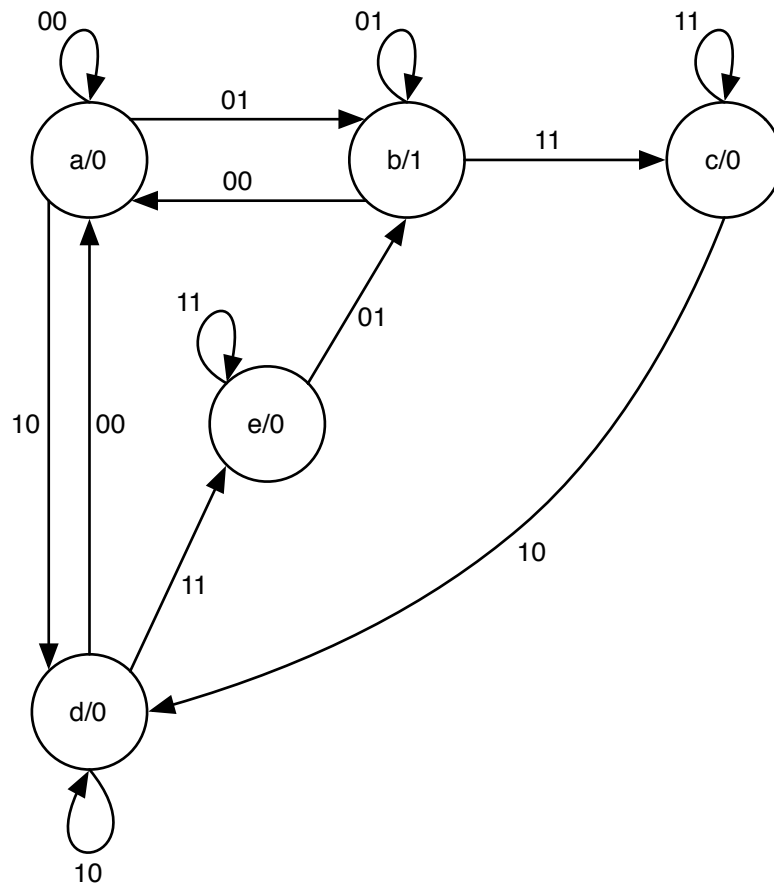


Abbildung 5.4.: Moore-Automat welcher reduziert werden soll

z	x_1x_0				y
	00	01	11	10	
a	a	b	*	c	0
b	a	b	d	*	0
c	a	*	e	c	0
d	*	*	d	c	0
e	*	b	e	*	1

Tabelle 5.1.: Tabelle zum Automaten aus Abbildung 5.4

5. Endliche Automaten zur Schaltwerksmodellierung

Aus der Tabelle 5.4 können die kompatiblen Zustände abgelesen werden. Dazu wird jede Zeile mit jeder Folgezeile auf Kompatibilität geprüft. a ist also in diesem Beispiel mit allen anderen Zuständen b , c , d und e kompatibel. Um eine Übersicht über alle kompatiblen Zustände zu erhalten wird das Kompatibilitätsdiagramm verwendet. Eine Verbindung von einem Zustand zu einem anderen existiert genau dann, wenn die beiden Zustände zueinander kompatibel sind. Abbildung 5.5 zeigt das Kompatibilitätsdiagramm zum besprochenen Beispiel.

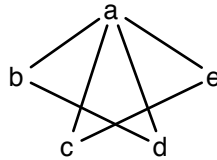


Abbildung 5.5.: Kompatibilitätsdiagramm zur Tabelle 5.4

Mit Hilfe des Diagramms sind Gruppen von vollständig kompatiblen Zuständen ersichtlich. Zur Gruppenbildung dürfen nur bisher unberücksichtigte Zustände verwendet werden. Für das Beispiel gilt hier:

$$\begin{aligned}
 a - c - e &\mapsto \{a, c, e\} \\
 a - b - d &\mapsto \{b, d\}
 \end{aligned}$$

Aus den Gruppen ergeben sich in diesem Beispiel zwei neue Zustände:

$$\begin{aligned}
 \{a, c, e\} &\mapsto a' \\
 \{b, d\} &\mapsto b'
 \end{aligned}$$

Die Ausgabefunktion bei dieser Wahl der Zustände ist allerdings nicht mehr mit einem Moore-Automaten realisierbar, da die Ausgabefunktion zusätzlich von der Eingabebelegung abhängt. Der Automat wird deshalb in einen Mealy-Automaten überführt. Tabelle 5.2 zeigt die tabellarische Ansicht des Mealy-Automaten.

z	x_1x_0			
	00	01	11	10
a'	$a'/0$	$b'/0$	$a'/1$	$a'/0$
b'	$a'/0$	$b'/0$	$b'/0$	$a'/0$

Tabelle 5.2.: Reduzierte Automatentabelle

6. Schaltwerksynthese

Wie aus einem Automaten ein Schaltwerk entsteht, wird in diesem Kapitel besprochen. Zunächst geht es um die Zustandscodierung, anschließend um die Generierung von der algebraischen Übergangs- und Ausgangsfunktionen und schließlich um die Hardwaresynthese.

6.1. Zustandscodierung

Den Zuständen des zu realisierenden Automats werden Bitvektoren mit k vielen Stellen zugewiesen. Jede Stelle eines Bitvektors wird durch ein Speicherelement realisiert. Als Speicherelemente können die in diesem Dokument vorgestellten JK- oder D-FlipFlops verwendet werden. Für n verschiedene Zustände werden demnach $k = \lceil \lg(n) \rceil$ viele Stellen benötigt.

Durch geschickte Wahl der Stellencodierung können die Schaltnetzkosten reduziert und die Ansteuerfunktionen der Speicherelemente vereinfacht werden. Dabei um dies zu erreichen ist es oft sinnvoll, dass die Stellencodierung eine Zuordnung zum Schaltwerksverhalten erkennen lässt.

Als Beispiel für eine Zustandscodierung wird der Binärzahl zu Zahl im Graycode Wandler aus Abbildung 5.1 verwendet. Der Automat hat 5 Zustände. Demnach müssen $k = \lceil \lg(5) \rceil = 3$ viele Stellen codiert werden. Die Codierung ist im Diagramm aus Abbildung 6.1 gezeigt.

6.2. Ansteuerfunktionen

Ansteuerfunktionen¹ sind Funktionen, welche die Eingabebelegung der Speicherbausteine eines Schaltwerks aus dem aktuellen Zustand und eventuellen Eingabedaten berechnen. Zur Generierung der Ansteuerfunktionen wird eine Zustandstabelle verwendet, welche alle codierten Zustände und die dazugehörigen Belegungen der Speicherbausteine enthält. Der erste Spaltenblock definiert den aktuellen Zustand. Der zweite Spaltenblock definiert die Eingabevariablen. Der dritte Spaltenblock definiert den Folgezustand. Der vierte Spaltenblock gibt die Belegung der Speicherbausteine an, womit der Zustand des ersten Spaltenblocks in den Zustand des dritten Spaltenblocks überführt werden kann. Der fünfte Spaltenblock gibt schließlich die Ausgabe je Zustands-Eingabe-Kombination an.

Tabelle 6.2 gibt die Zustandstabelle für den Binär-Gray-Code-Wandler an.

¹Ansteuerfunktionen werden auch Überföhrungsfunktionen genannt.

6. Schaltungssynthese

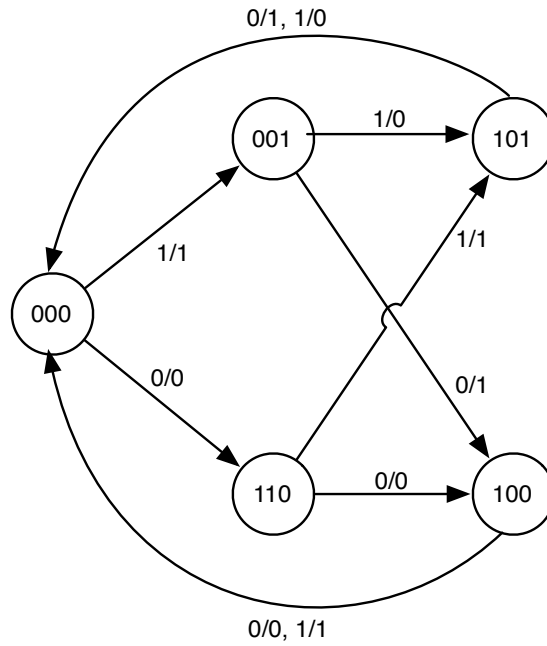


Abbildung 6.1.: Beispiel zur Zustandskodierung vom Automaten aus Abbildung 5.1

z^t					z^{t+1}							
z_2	z_1	z_0	x		z_2	z_1	z_0	J_2K_2	J_1K_1	J_0K_0	y^t	
0	0	0	0		1	1	0	1 *	1 *	0 *	0	
0	0	0	1		0	0	1	0 *	0 *	1 *	1	
0	0	1	0		1	0	0	1 *	0 *	* 1	1	
0	0	1	1		1	0	1	1 *	0 *	* 0	0	
0	1	0	0		*	*	*	* *	* *	* *	*	
0	1	0	1		*	*	*	* *	* *	* *	*	
0	1	1	0		*	*	*	* *	* *	* *	*	
0	1	1	1		*	*	*	* *	* *	* *	*	
1	0	0	0		0	0	0	* 1	0 *	0 *	0	
1	0	0	1		0	0	0	* 1	0 *	0 *	1	
1	0	1	0		0	0	0	* 1	0 *	0 *	1	
1	0	1	1		0	0	0	* 1	0 *	0 *	0	
1	1	0	0		1	0	0	* 0	* 1	* 1	0	
1	1	0	1		1	0	1	* 0	* 1	* 1	1	
1	1	1	0		*	*	*	* *	* *	* *	*	
1	1	1	1		*	*	*	* *	* *	* *	*	

Tabelle 6.1.: Zustandstabelle zum Automaten aus Abbildung 6.1

6. Schaltwerksynthese

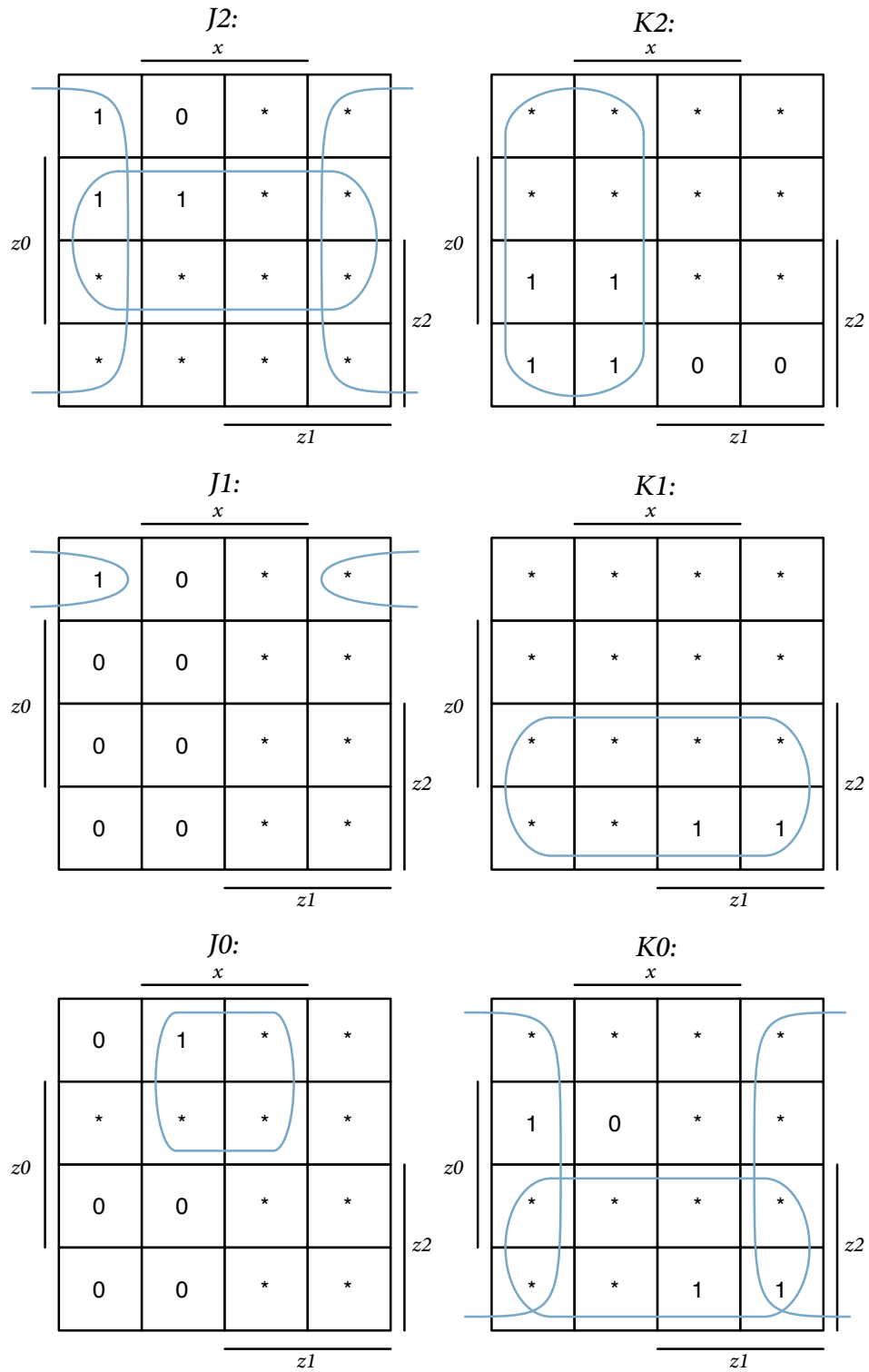


Abbildung 6.2.: KV-Diagramme für die Ansteuerfunktionen aus Tabelle 6.2

6. Schaltwerksynthese

In der Zustandstabelle 6.2 ist zu sehen, wie die JK-FlipFlops je Stelle beschaltet werden müssen um einen Zustandswechsel zu erzeugen. Beispielsweise in der ersten Zeile wechselt die Stelle $z_2^t = 0$ bei Eingabe von $x = 0$ auf $z_2^{t+1} = 1$. Um diesen Wechsel zu erzeugen ist das J_2K_2 -FlipFlop verantwortlich. Ein Wechsel von 0 auf 1 wird mit der Belegung $J_2 = 1$ und $K_2 = *$ erreicht. Für die weiteren Stellen z_1 und z_0 sind die jeweiligen FlipFlops J_1K_1 und J_0K_0 zuständig.

In diesem Fall hängt der Folgezustand z^{t+1} vom vorhergehenden Zustand z^t und der Eingabe x ab. Daher müssen die Ansteuerfunktionen in Abhängigkeit des vorhergehenden Zustands z^t und der Eingabe x generiert werden. Es wird für jedes J_i und K_i eine Ansteuerfunktion angegeben. Abbildung 6.2 zeigt die KV-Diagramme, welche zur Minimierung der Ansteuerfunktionen verwendet werden. Die Ansteuerfunktionen sind:

$$\begin{aligned}J_2 &= z_0 + \bar{x} \\K_2 &= \bar{z}_1 \\J_1 &= \bar{x} \bar{z}_0 \bar{z}_2 \\K_1 &= z_2 \\J_0 &= x \bar{z}_2 \\K_0 &= \bar{x} + z_2\end{aligned}$$

Notiz: da immer entweder J oder K „*“ ist, können die Ansteuerfunktionen von J und K zu einer Funktion zusammengefasst werden. Die KV-Diagramme für J bzw. K können somit verschmolzen und daraus eine einzige Ansteuerfunktion für J und K gewonnen werden.

7. Ausgewählte Schaltwerke

In folgenden Abschnitten wird besprochen, wie sich Speicherlemente zu Registern erweitern lassen. Außerdem werden typische Zählerschaltungen vorgestellt.

7.1. Register

Register dienen der Speicherung von mehreren Bits. Man spricht hier von Wörtern bestehend aus einer Folge von Bits. Mit Hilfe von Spezialregistern können auch arithmetische und logische Operationen auf Bitwörtern durchgeführt werden.

7.1.1. Parallelregister

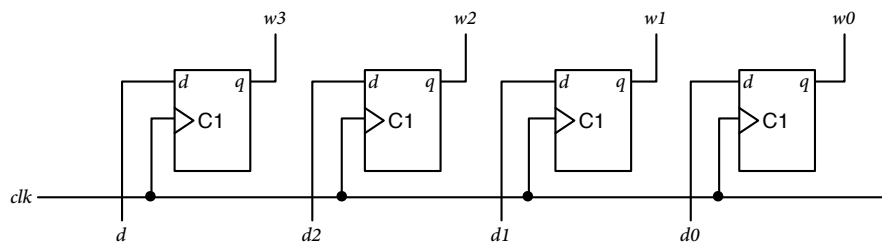


Abbildung 7.1.: Strukturdiagramm eines 4-Bit-Parallelregisters

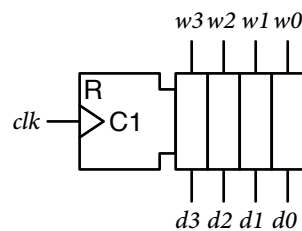


Abbildung 7.2.: Schaltsymbol eines 4-Bit-Parallelregisters

Das Parallelregister ist von der Komplexität her das einfachste unter den Register-typen. Für jedes Bit, das gespeichert werden soll, wird hier ein D-FlipFlop verwendet. Die Anzahl an verwendeten D-FlipFlops gibt die Wortlänge der Wörter an, welche im Register gespeichert werden kann. In einem Parallelregister mit der Wortlängenkapazität von n -Bits sind n -viele D-FlipFlops parallel geschaltet und es existieren n -viele

parallele Dateneingänge und n -viele parallele Datenausgänge. Abbildung 7.1 zeigt das Strukturbild eines 4-Bit-Parallelregisters. Abbildung 7.2 zeigt das normierte Schaltbild des 4-Bit-Parallelregisters.

7.1.2. Schieberegister

Ein Schieberegister wird im Gegensatz zum Parallelregister sequentiell Bit für Bit befüllt. Das Datenwort kann wiederum parallel über die Ausgänge der D-FlipFlops ausgelesen werden. Abbildung 7.3 zeigt die Struktur eines 4-Bit-Schieberegisters. Der Ausgang jedes D-FlipFlops wird nach Außen zum Auslesen geführt und zusätzlich zum Eingang des darauffolgenden D-FlipFlops geführt um die Schiebeoperation zu realisieren. Abbildung 7.4 zeigt das Schaltsymbol eines 4-Bit-Schieberegisters.

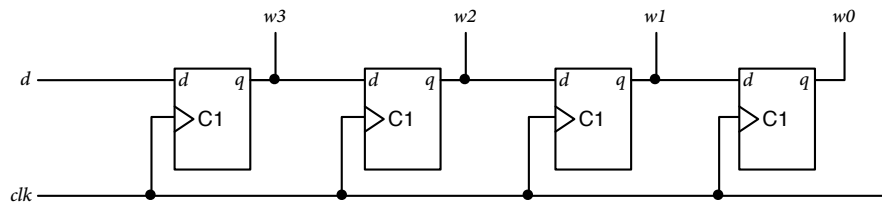


Abbildung 7.3.: Strukturdiagramm eines 4-Bit-Rechts-Schieberegisters

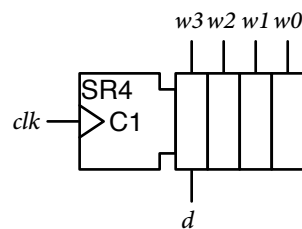


Abbildung 7.4.: Schaltsymbol eines 4-Bit-Rechts-Schieberegisters

Das Schieberegister kann zur Parallelisierung eines seriellen Datenstroms genutzt werden. Dazu werden die Bits des Datenstroms Bit für Bit in das Register geschrieben und jedes mal, wenn das Register vollständig beschrieben wurde werden die gespeicherten Bits an die Ausgänge weitergeleitet. Sind n parallele Eingänge und ein Ausgang gegeben ist somit auch eine Serialisierung eines parallelen Datenstroms möglich.

Außerdem können Schieberegister zur Manipulation der Bits verwendet werden um arithmetische und logische Bitschiebe-Operationen hardwaretechnisch umzusetzen. Ein Links-Schieben um n -Stellen, wobei die jeweils niedrigste Stelle mit 0 aufgefüllt wird, bewirkt bei Interpretation der Bitfolge als Dualzahl eine n -fache Verdoppelung des Wertes. Dies ist selbstverständlich nur dann korrekt, wenn durch das Schieben nicht signifikante Bits verloren werden. Analog bewirkt ein Rechts-Schieben um n -Stellen eine n -fache

7. Ausgewählte Schaltwerke

Halbierung des Wertes. Exakt gilt dies natürlich wiederum nur dann, wenn die niederwertigste Stelle mit 0 belegt war. Ansonsten entspricht die Rechts-Schiebung immer der n -fachen Halbierung abgerundet.

Zusätzlich ist es möglich ein Ringschiebe-Register zu erstellen, indem der der Ausgang des letzten FlipFlops mit dem Eingang des ersten FlipFlops verbunden wird. Abbildung 7.5 veranschaulicht die Konstruktion anhand des Schaltsymbols.

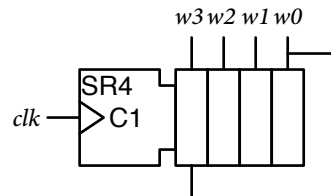


Abbildung 7.5.: Schaltsymbol eines 4-Bit-Ringschiebe-Registers

Es kann auch ein Links-Rechts-Schiebe-Register realisiert werden indem die jeweilige Funktion mittels eines Steuersignals ausgewählt wird. Das Steuersignal wird von Multiplexern verarbeitet, die abhängig vom Steuersignal den Signalausgang des linken oder rechten Speicherelements zur Eingabe in das nachfolgende Speicherelement wählen. Ein Schaltdiagramm, welches ein solches Register realisiert ist in Abbildung 7.6 zu sehen. Abbildung 7.7 zeigt das Schaltsymbol eines Links-Rechts-Schieberegisters.

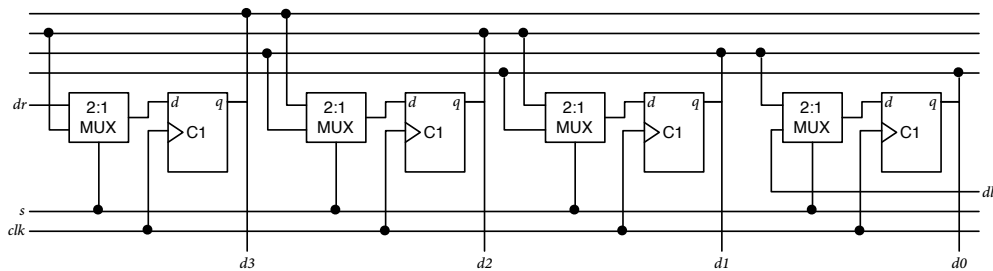


Abbildung 7.6.: Strukturdiagramm eines 4-Bit-Links-Rechts-Schieberegisters

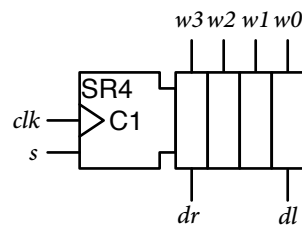


Abbildung 7.7.: Schaltsymbol eines 4-Bit-Links-Rechts-Schiebe-Registers

Je nachdem was mit dem Schieberegister realisiert werden soll müssen die Eingänge

d_r und d_l unterschiedlich belegt werden. Soll eine arithmetische Linksschiebung erfolgen, was einer Multiplikation mit der entsprechenden Zweierpotenz gleich ist, müssen 0en am Eingang d_l angelegt werden und das Steuersignal s auf op-code Linksschieben gesetzt werden. Soll eine arithmetische Rechtsschiebung erfolgen, was einer Division mit der entsprechenden Zweierpotenz gleich ist, muss der Wert des Vorzeichen-Bits an d_r angelegt und der das Steuersignal s auf den op-code Rechtsschieben gesetzt werden. Handelt es sich um eine unsigned Zahlendarstellung muss natürlich entsprechend mit 0en in beiden Fällen aufgefüllt werden.

7.2. Zählerschaltungen

Zählschaltungen zählen elektrische Impulse. Realisiert wird ein Zähler durch logische Vernetzung von FlipFlops. Die FlipFlops dienen der Speicherung des Zählerstandes um beim nächsten Signal weiter zählen zu können. Ein Zustand entspricht dabei einem Zahlensymbol. Es wird zwischen synchronen und asynchronen Zählern unterschieden.

7.2.1. Synchrone Zähler

Bei synchronen Zählern liegt der Zählimpuls oder Takt an allen FlipFlops gleichzeitig an. Dadurch schalten alle FlipFlops gleichzeitig und die Ausgangsleitungen halten synchron den korrekten Wert. Bei den synchronen Zählern wird zwischen Taktgesteuerten und Signalgesteuerten Zählern unterschieden.

Taktgesteuerte Zähler basieren auf den besprochenen Verfahren um Schaltwerke zu konstruieren. Jedes FlipFlop erhält das gleiche Taktsignal zur selben Zeit. Die Eingänge der FlipFlops werden anhand der Überföhrungsfunktion beschaltet. Der Zähler zählt dabei automatisch zu jedem Zeitpunkt und benötigt deshalb normalerweise keine weiteren zusätzlichen Eingabeparameter abgesehen vom momentanen Zustand.

Als Beispiel sei hier ein 5-stelliger Johnson-Zähler gegeben. Der Zähler besitzt 10 Zählerstellungen, die in Abbildung 7.8 im Grafen abgebildet sind. Tabelle 7.1 zeigt die Zustandstabelle zum Johnson-Zähler. Aus der Tabelle lassen sich die Ansteuerfunktionen für die einzelnen FlipFlops direkt ablesen:

$$\begin{aligned} J_4 &= K_4 = \overline{q_4}q_3q_2q_1q_0 + q_4\overline{q_3}\overline{q_2}\overline{q_1}\overline{q_0} \\ J_3 &= K_3 = \overline{q_4}\overline{q_3}q_2q_1q_0 + q_4q_3\overline{q_2}\overline{q_1}\overline{q_0} \\ J_2 &= K_2 = \overline{q_4}\overline{q_3}\overline{q_2}q_1q_0 + q_4q_3q_2\overline{q_1}\overline{q_0} \\ J_1 &= K_1 = \overline{q_4}\overline{q_3}\overline{q_2}\overline{q_1}q_0 + q_4q_3q_2q_1\overline{q_0} \\ J_0 &= K_0 = \overline{q_4}\overline{q_3}\overline{q_2}\overline{q_1}\overline{q_0} + q_4q_3q_2q_1q_0 \end{aligned}$$

Bei genauerem Hinsehen fällt auf, dass die Spalten der JK-Belegung sich decken mit den Spalten der q_i^t Belegung. Demnach können die Überföhrungsfunktionen auch ganz

7. Ausgewählte Schaltwerke

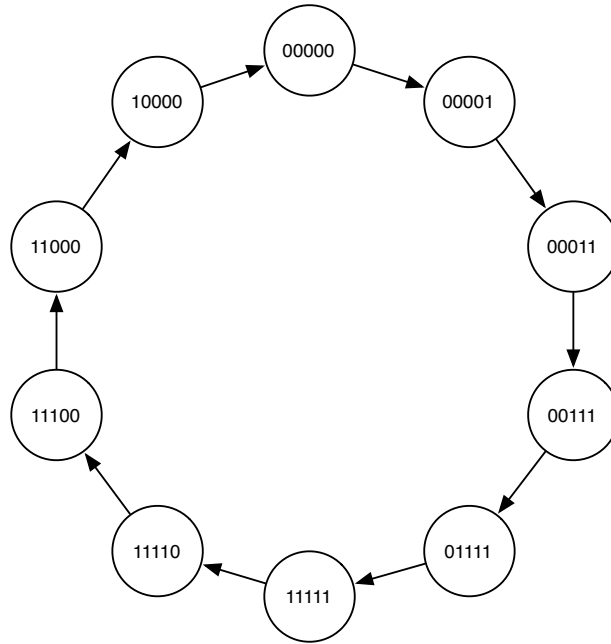


Abbildung 7.8.: Graf des Johnson-Zählers

z^t		z^{t+1}		FlipFlop-Vorbereitungseingänge				
$q_4 q_3 q_2 q_1 q_0$	$q_4 q_3 q_2 q_1 q_0$	$q_4 q_3 q_2 q_1 q_0$	$q_4 q_3 q_2 q_1 q_0$	$J_4 K_4$	$J_3 K_3$	$J_2 K_2$	$J_1 K_1$	$J_0 K_0$
00000	00001	00000	00001	0*	0*	*0	0*	1*
00001	00011	00001	00011	0*	0*	*0	1*	*0
00011	00111	00011	00111	0*	0*	1*	*0	*0
00111	01111	00111	01111	0*	1*	*0	*0	*0
01111	11111	01111	11111	1*	*0	*0	*0	*0
11111	11110	11111	11110	*0	*0	*0	*0	*1
11110	11100	11110	11100	*0	*0	*0	*1	0*
11100	11000	11100	11000	*0	*0	*1	0*	0*
11000	10000	11000	10000	*0	*1	*0	0*	0*
10000	00000	10000	00000	*1	0*	*0	0*	0*

Tabelle 7.1.: Zustandstabelle des Johnson-Zählers

7. Ausgewählte Schaltwerke

einfach folgendermaßen festgelegt werden:

$$J_4 = q_3$$

$$K_4 = \overline{q_3}$$

$$J_3 = q_2$$

$$K_3 = \overline{q_2}$$

$$J_2 = q_1$$

$$K_2 = \overline{q_1}$$

$$J_1 = q_0$$

$$K_1 = \overline{q_0}$$

$$J_0 = \overline{q_4}$$

$$K_0 = q_4$$

Damit ergibt sich eine sehr anschauliche Implementierung wie in Abbildung 7.9 zu sehen.

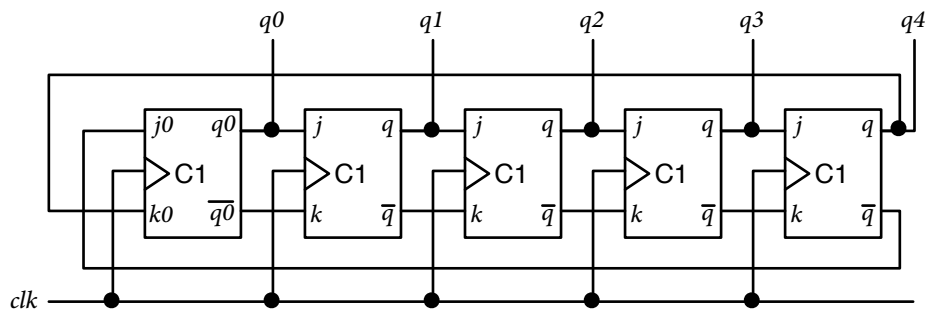


Abbildung 7.9.: Implementierung des Johnson-Zählers

Bei signalgesteuerten Zählern wird das Taktsignal so manipuliert, dass das Taktsignal nur dann anliegt, wenn das FlipFlop kippen soll. Alle Signaleingänge werden so beschaltet, dass das FlipFlop kippt.

7.2.2. Asynchrone Zähler

Bei asynchronen Zählern werden die jeweiligen nachfolgenden FlipFlops, welche die höheren Stellen repräsentieren, von den Vorgänger jeweiligen Vorgänger-FlipFlops getaktet. Ein FlipFlop schält also immer dann um, wenn das VorgängerFlipFlop von 0 auf 1 umschält.

Tabelle 7.2 zeigt die Synthesetabelle für einen 8-4-2-1-BCD-Zähler. Der Tabelle kann folgendes Taktschema entnommen werden:

- das zweite FlipFlop wird vom $\overline{q_0}$ Ausgang des ersten FlipFlops getaktet.

7. Ausgewählte Schaltwerke

- das dritte FlipFlop wird vom $\overline{q_1}$ Ausgang des ersten FlipFlops getaktet.
- das vierte FlipFlop wird vom $\overline{q_2}$ Ausgang des dritten FlipFlops getaktet.

Die Ansteuerfunktionen sind laut Synthesetabelle 7.2:

$$\begin{aligned} J_3 &= q_2 q_1 \\ K_3 &= 1 \\ J_2 &= 1 \\ K_2 &= 1 \\ J_1 &= \overline{q_3} \\ K_1 &= 1 \\ J_0 &= 1 \\ K_0 &= 1 \end{aligned}$$

Abbildung 7.10 zeigt das Schaltdiagramm des 8-4-2-1-BCD-Zählers.

z^t	z^{t+1}	FlipFlop-Vorbereitungseingänge			
$q_3 q_2 q_1 q_0$	$q_3 q_2 q_1 q_0$	$J_3 K_3$	$J_2 K_2$	$J_1 K_1$	$J_0 K_0$
0000	0001	**	**	**	1*
0001	0010	0*	**	1*	*1
0010	0011	**	**	**	1*
0011	0100	0*	1*	1*	*1
0100	0101	**	**	**	1*
0101	0110	0*	**	1*	*1
0110	0111	**	**	**	1*
0111	1000	1*	1*	1*	*1
1000	1001	**	**	**	1*
1001	0000	*1	**	0*	*1

Tabelle 7.2.: Synthesetabelle für einen asynchronen 8-4-2-1-BCD-Zähler

7.3. Speicher

In digitalen Rechenanlagen werden verschiedene arten von Speicher verwendet. Besonders geläufig sind wohl der Arbeitsspeicher¹ und der Stack. Hier wird weiterhin der Pufferspeicher, auch als Queue bezeichnet besprochen.

Der Arbeitsspeicher besteht aus einem Speicher von 2^k -vielen Bit-Wörtern, die mit k -vielen Adressleitungen angesprochen werden können. Jedes Speicherwort speichert dabei n -viele Bits. Insgesamt können also $2^k \cdot n$ viele Bits gespeichert werden.

¹Arbeitsspeicher wird auch als Random Access Memory bezeichnet (RAM)

7. Ausgewählte Schaltwerke

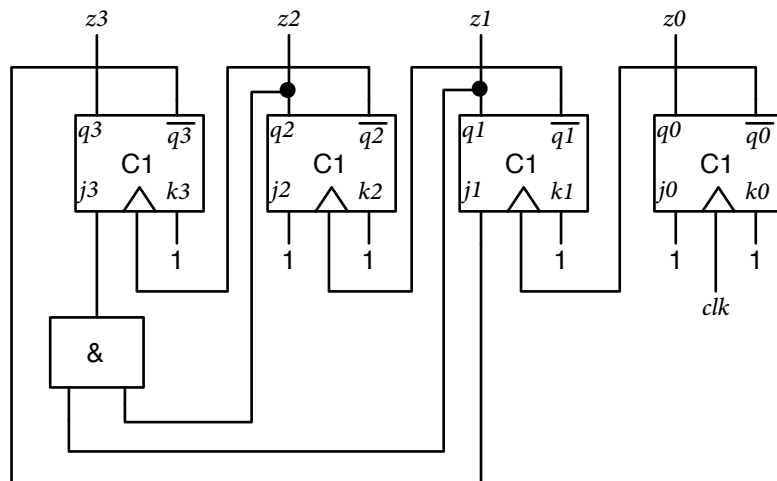


Abbildung 7.10.: Schaltdiagramm für einen asynchronen 8-4-2-1-BCD-Zähler

Der Pufferspeicher ist ein Speicher nach dem Prinzip „First in First Out“, auch kurz „FiFo“ genannt. Es wird zum Speichern immer an letzter Position ein Speicherwort angehängt und beim lesen immer das Speicherwort an erster Position gelesen. Somit wird immer das Speicherwort gelesen, welches als erstes in den Pufferspeicher gelegt wurde. Um dieses Verhalten zu realisieren wird ein Steuerwerk vor die Register geschaltet. Das Steuerwerk gibt zusätzlich zwei Signale aus. Das eine Signal signalisiert, ob der Speicher voll ist. Das andere Signal signalisiert, ob der Speicher leer ist.

Der Stack funktioniert nach dem „Last in First Out“ oder kurz „LiFo“ Prinzip. Dies bedeutet, dass das zuletzt hinzugefügte Speicherwort auch das erste ist, welches wieder gelesen wird. Die Steuerung wird wiederum einem spezifischen Steuerwerk überlassen, welches wiederum Signale für voll und leer nach außen weitergibt.