Klassische Gatter und Logikelemente

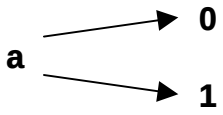
Seminarvortrag zu
Ausgewählte Kapitel der Quantentheorie
„Quantenalgorithmen“

Gerd Ch. Krizek
WS 2003

I. Grundlagen und Methoden der Logik:

Im folgenden soll die Konstruktion und anschließende Implementation von logischen Systemen besprochen werden. Dabei wird nur die binäre Logik betrachtet.

Logische Variable:
In einer binären Logik ordnet man einer logischen Variablen **a** zwei mögliche Werte zu:



Logische Funktion:
Logische Variablen können nun miteinander verknüpft werden. Diese Verknüpfung erfolgt mit Hilfe der elementaren logischen Funktionen. Die elementaren logischen Funktionen werden im Kapitel II noch ausführlich erläutert. Eine logische Funktion kann man nun in folgender Form schreiben:

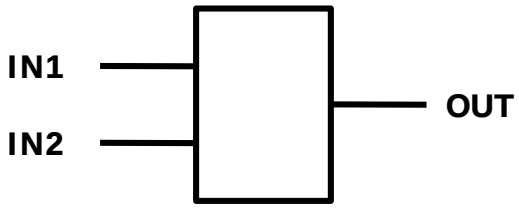
$$Y = (X_1 \wedge X_2) \vee (X_3 \wedge \overline{X_4})$$

Schreibt man eine logische Funktion in dieser Form auf spricht man von einer Boolschen Gleichung.

Die Logik verfügt über verschiedene Methoden die zur Entwicklung von logischen Systemen zur Verfügung stehen. Vorab sollen diese Methoden erläutert werden:

I.1 Wahrheitstabelle:

Die Wahrheitstabelle verknüpft logische Eingänge zu einem oder mehreren logischen Ausgängen. Die logischen Ausgänge können nach belieben gewählt werden und charakterisieren damit das Verhalten des logischen Systems.



IN1	IN2	OUT
0	0	X_1
0	1	X_2
1	0	X_3
1	1	X_4

Die Wahrheitstabelle repräsentiert die logische Funktion, durch die logische Eingänge mit logischen Ausgängen verknüpft ist. Die Zahl der Eingänge legt die Anzahl der Zeilen der Wahrheitstabelle fest, da alle möglichen Eingangszustände definiert sein müssen.

So ist es nicht sinnvoll folgende Wahrheitstabelle für obiges System aufzustellen,

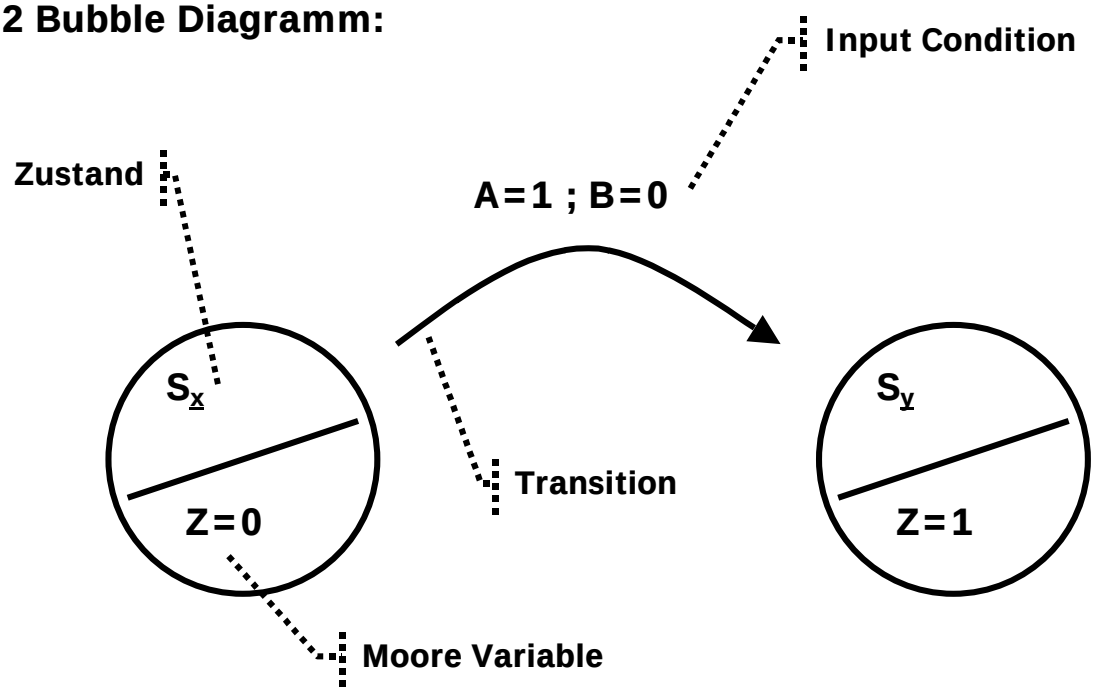
IN1	IN2	OUT
0	0	X_1
0	1	X_2
1	1	X_3

da eine logische Eingangskombination nicht definiert ist. Wie wir später sehen werden, ist es aber notwendig eine vollständige Abdeckung aller Zustände des Systems zu realisieren.

Wahrheitstabellen können im Ausgang beliebige Breite haben. Es müssen nicht alle logischen Kombinationen von Ausgängen abgedeckt sein. So könnte auch folgende Wahrheitstabelle auftreten.

IN1	IN2	OUT1	OUT2
0	0	X_1	Y_1
0	1	X_2	Y_2
1	0	X_3	Y_3
1	1	X_4	Y_4

I.2 Bubble Diagramm:



Jeder Zustand geht bei einer Input Condition in einen anderen Zustand des logischen Systems über. Die jeweilige Übergangskondition steht bei der „Transition" dabei. In jedem Zustandsbubble steht der jeweilige logische Ausgang des Systems. Bubble Diagramms werden verwendet, um sequentielle Logiksysteme zu entwerfen.

II. Kombinatorische Logik

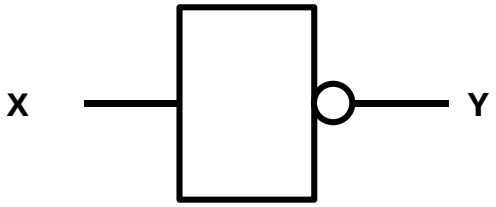
Kombinatorische Logik ist eine zeitlich nicht gesteuerte logische Verknüpfung, die mit logischen Eingängen und einer logischen Funktion einen oder mehrere logischen Ausgang liefert. So liefert die theoretische kombinatorische Logik bei definierten logischen Eingängen instantan einen oder mehrere logische Ausgänge.

Bei einer realistischen kombinatorischen Logik treten massive Probleme auf, da die Laufzeiten durch die Gatter (wie wir die Realisierung der logischen Funktion von nun an nennen wollen) durchwegs nicht gleichartig sein müssen. Dies führt auf das Problem der Hazards auf das ich später noch einmal zurückkommen werde.

II.1 Elementare Logikfunktionen:

Jede logische Funktion ist mit Hilfe der elementaren Logikfunktionen realisierbar. In Folge sollen diese einfachsten Logikfunktionen vorgestellt werden.

Inverter:

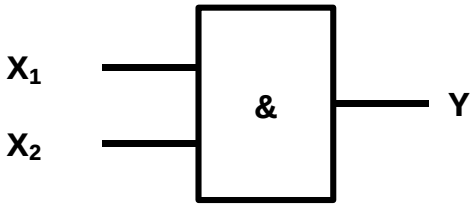


$$Y = \overline{X}$$

X	Y
0	1
1	0

Der Inverter liefert am Ausgang einfach das Inverse des logischen Eingangs.

AND:

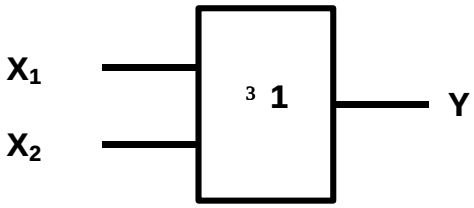


$$Y = X_1 \wedge X_2$$

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Das AND ist mit der Multiplikation eng verwandt, und liefert nur dann eine logische 1 wenn beide Eingänge 1 sind.

OR:

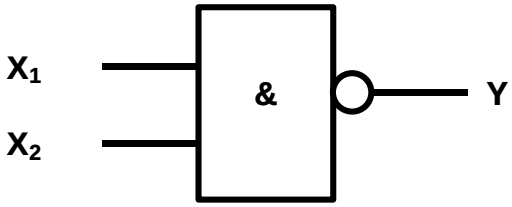


$$Y = X_1 \vee X_2$$

X_1	X_2	**Y**
0	0	0
0	1	1
1	0	1
1	1	1

Das OR ist der Addition sehr ähnlich und liefert dann eine logische 1 wenn einer der beiden Eingänge auf logisch 1 ist.

NAND:

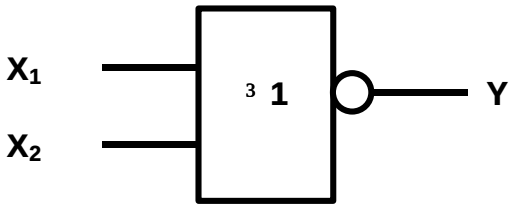


$$Y = \overline{X_1 \wedge X_2}$$

X_1	X_2	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

Das NAND ist ein AND mit einer Inversion des Ausganges.

NOR:



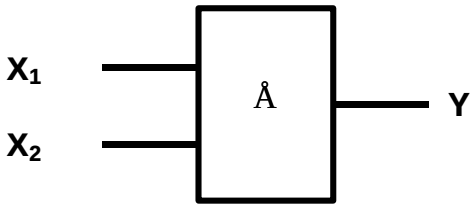
$$Y = \overline{X_1 \vee X_2}$$

X_1	X_2	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

Das NOR ist ein OR mit einer Inversion des Ausganges.

XOR:

$$Y = X_1 \oplus X_2$$



X_1	X_2	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

Das XOR oder „EXKLUSIVE OR" liefert nur bei einer logischen 1 an einem Eingang eine logische 1 am Ausgang

11.2 Komplexe Logikfunktionen:

Alle komplexen Logikfunktionen sind mit den elementaren Logikfunktionen realisierbar. Beispiele solcher komplexer Logikfunktionen sind:

- Multiplexer
- Demultiplexer
- Decoder
- Priority-encoder
- Komparator

Wir betrachten jetzt folgende Logikfunktion, die wir gerne mit Hilfe der elementaren Logikfunktionen darstellen wollen.

A_0	A_1	B_0	B_1	**OUT1**	**OUT2**
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	1	0

Ich werde das Verfahren an diesem Beispiel demonstrieren, aber nicht systematisch erklären.

Wir werden jetzt in Folge die Logikfunktion der gesuchten kombinatorischen Logik ermitteln. Dazu müssen wir jeden der beiden Ausgänge getrennt behandeln.

Zuerst betrachten wir OUT1:

1. Schritt: Bilden des ON-Sets

Das ON-Set sind alle Ausgänge die eine logische 1 liefern.

A_0	A_1	B_0	B_1	**OUT1**
0	1	1	1	1
1	1	1	1	1

2. Schritt: Mintermbildung:

Die Minterme sind die einfachste logische Funktion die eine Ausgangsfunktion beschreibt. Man invertiert jeden logischen Eingang der 0 ist und verknüpft alle Eingänge mit AND.

$$m_7 = \overline{A_0} \wedge A_1 \wedge B_0 \wedge B_1$$
$$m_{15} = A_0 \wedge A_1 \wedge B_0 \wedge B_1$$

3. Schritt: Funktionsbildung:

Die logische Funktion der gesamten kombinatorischen Logik ist nun durch die Verknüpfung von OR zwischen allen Mintermen gegeben. In unserem Beispiel ist dies relativ einfach:

$$y_{OUT1} = \left(\overline{A_0} \wedge A_1 \wedge B_0 \wedge B_1\right) \vee \left(A_0 \wedge A_1 \wedge B_0 \wedge B_1\right)$$

Für den Ausgang OUT 2 ist es ganz analog und die Logikfunktion ergibt sich zu:

$$y_{OUT2} = \left(\overline{A_0} \wedge A_1 \wedge B_0 \wedge \overline{B_1}\right) \vee \left(A_0 \wedge A_1 \wedge B_0 \wedge \overline{B_1}\right)$$

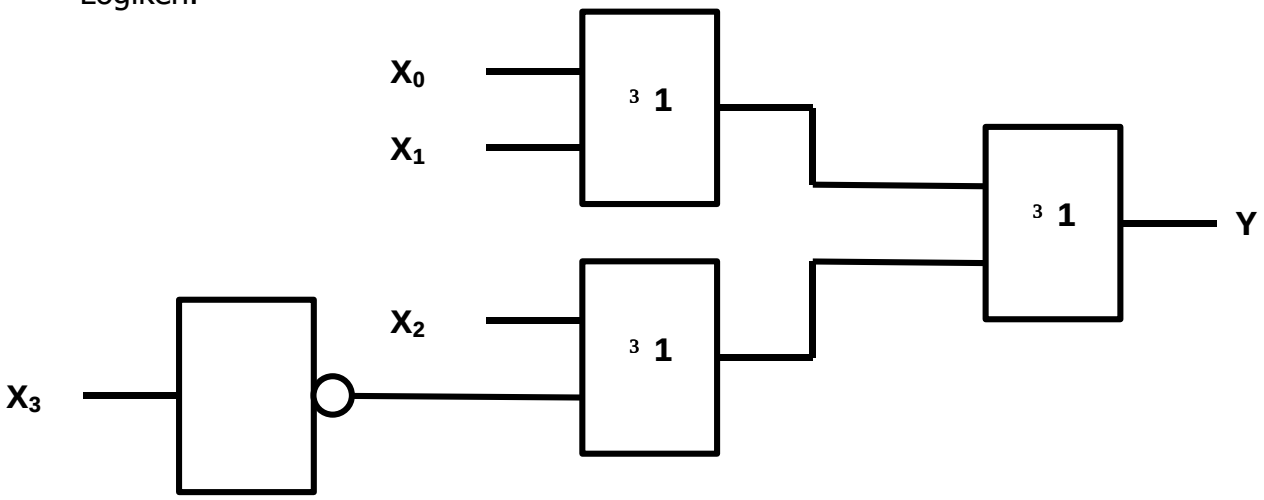
Diese zwei Logikfunktionen sind nun eine vollständige Beschreibung der kombinatorischen Logik. Die Funkionen sind aber nur deshalb so einfach, will wir ein relativ kleines ON-Set für das Beispiel gewählt haben. Bei größeren ON-Sets ist die Funktion meist sehr unübersichtlich.

Es ist aber möglich eine kleinere Form logischen Funktion zu finden. Diese ist durch Minimierung im KV-Diagramm zu erreichen. Dies führt aber in unserem Rahmen zu weit.

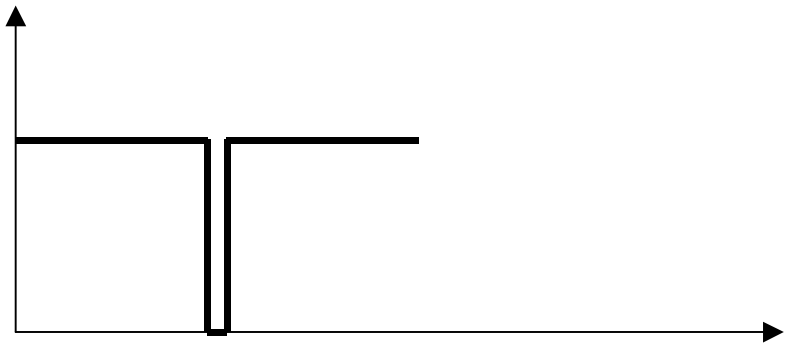
III. Sequentielle Logik:

Bis jetzt haben wir nur Logiken betrachtet, die zumindest theoretisch instantane logische Ausgänge bilden. Praktisch sind jedoch die Laufzeiten in den Gattern zu berücksichtigen. Diese führen auf die Hazardproblematik.

Ein Hazard ist eine ungewünschte Änderung des Ausganges bei kombinatorischen Logiken.

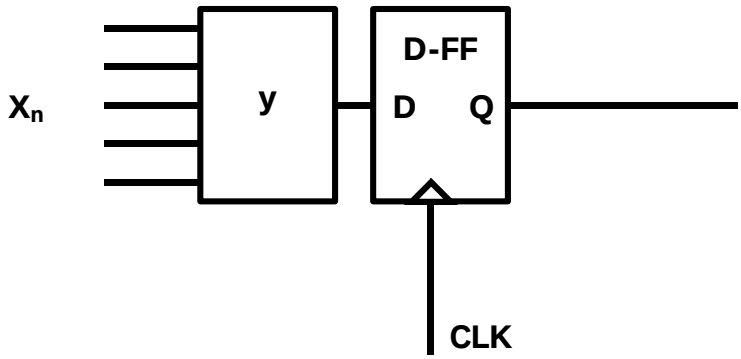


Werden alle Eingänge gleichzeitig angelegt, so ist der Ausgang X_3 mit der Verzögerungszeit des Inverters erst später an der OR-Verknüpfung. Dies kann zu einer Zustandsänderung führen, die aber bei Eintreffen des logischen Einganges vom Inverter wieder wechselt. Dies kann z.B. zu solchem Zeitverhalten am Ausgang führen.



Dieses ungeplante und nur von inneren physikalischen Bedingung abhängige Verhalten bezeichnet man als Hazard. Es gibt Implementierungsverfahren, bei denen Hazards vermieden werden können. Eine einfachere Methode ist die Pufferung mit zeitabhängigen Elementen. Dies führt uns auf die Sequentielle Logik.

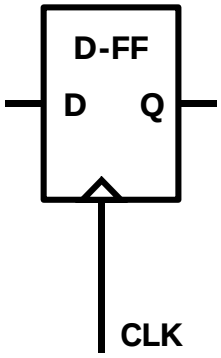
Pufferung von kombinatorischen Logiken mit D-Flip Flops kann die Hazards am Ausgang vermeiden. Dabei wird ein Dateneingang zu einem bestimmten Zeitpunkt, der durch den CLK Eingang gegeben ist übernommen.



III.1 Elementare sequentielle Logikelemente:

D-FlipFlop:

Das D-Flip Flop ist ein zeitgesteuertes Element, daß bei einer steigenden Flanke am CLK Eingang den logischen Wert des D Einganges (Daten Eingang) übernimmt und am Q Ausgang wiedergibt



D	Q_{ALT}	Q_{NEU}
0	X	0
1	X	1

Der Steuereingang CLK ist der globale Clock der bei großen Logiken die Steuerung aller Elemente übernimmt.

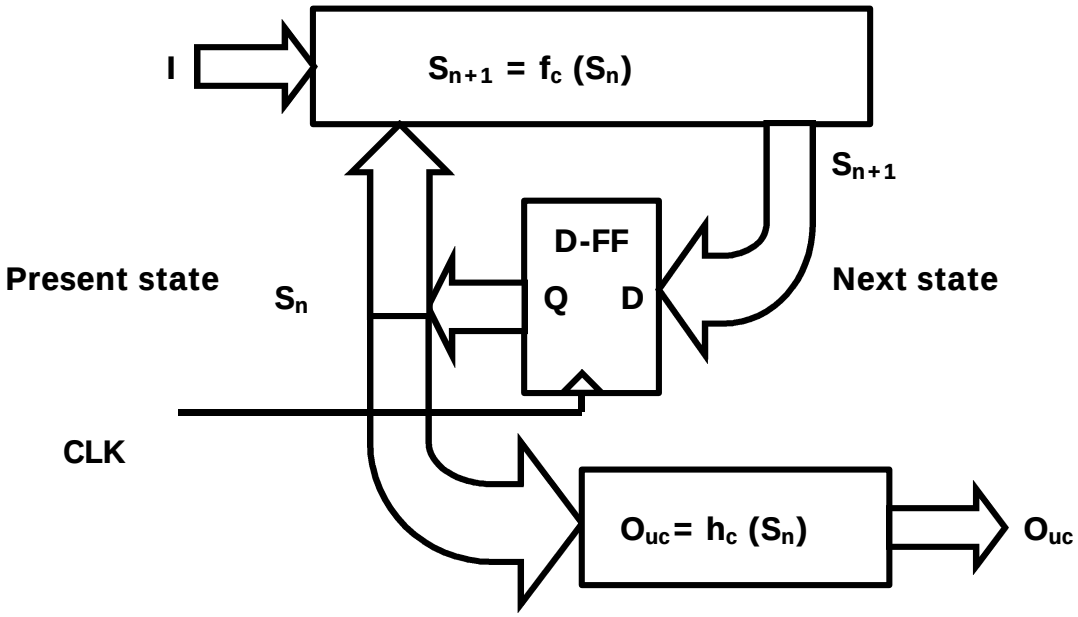
Folgende Flip Flop Arten gibt es ebenfalls, sind aber im Rahmen dieser Arbeit nicht von Interesse für uns:

JK-FlipFlop

T-FlipFlop

III.2 Finite state machines (FSM)

Finite state machines sind Zustandsmaschinen die von einem globalen Takt gesteuert werden. Wichtigstes Element ist das D-FF, das den aktuellen Zustand der FSM definiert.



Die FSM verfügt über Eingänge I, Ausgänge O_{uc} und einen globalen Steuertakt CLK. Die Eingänge und der aktuelle Zustand der FSM bestimmen den Zustand den die FSM als nächstes Annehmen soll. Von dem aktuellen Zustand werden Ausgänge mit einer kombinatorischen Logik gebildet, die die gewünschte Funktion generieren.

Die Beschreibung und Entwicklung der FSM erfolgt im Bubble Diagramm. Aus dem Bubble Diagramm werden dann die zwei kombinatorischen Logikfunktionen entnommen.

Obige FSM ist eine sogenannte „Moore-machine" ein Spezialfall der sogenannten allgemeinen FSM. Ein weiterer Spezialtyp ist die Mealy machine. Dort werden die Eingänge I mit einer zusätzlichen kombinatorischen Logik als Mealy Ausgänge der FSM benutzt.

Eine weitere Abwandlung der FSM ist die asynchrone state machine. Bei diesem Konzept wird auf den globalen Steuertakt verzichtet. Die Rückkopplung des „present state" in den „next state" erfolgt also nur durch eine kombinatorische Verknüpfung. Bei dieser Form der state machine muß sehr auf eine hazardfreie Implementierung der kombinatorischen Blöcke geachtet werden. Für weitere Informationen dazu verweise ich auf die weiterführende Literatur.

IV. Implementierung:

Wir haben jetzt die theoretische Behandlung von Logiksystemen besprochen. Jetzt bleibt noch offen, wie die logischen Variablen „in Hardware" realisiert sind.

Die Hardware-Realisierung von logischen Systemen erfolgt auf Basis von elektromagnetischen Größen. So wird der logische Zustand durch die Spannung am Eingang eines logischen Elementes realisiert. Man unterscheidet nun auch verschiedene Logikfamilien mit Spannungen.

IV.1 Logikfamilien:

TTL (Transistor-Transistor Logik):

Bei TTL werden die logischen Zustände durch die zwei Spannungen 0V und 5V realisiert.

CMOS:

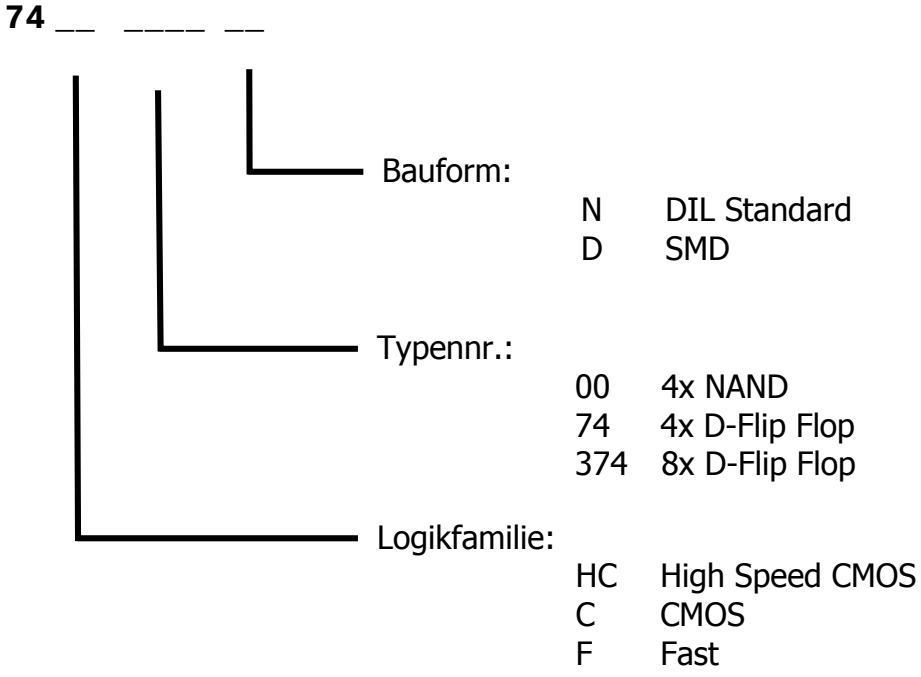
Bei CMOS ist die Wahl der Spannungen dynamischer und geht von 5 bis 15V.

Für Details verweise ich auf weiterführende Literatur, vor allem auf passende Datenbücher.

IV.2 Logikelemente:

74er Serie:

Die 74er Serie ist eine Logikserie in der verschiedene Formen der Standardelemente realisiert sind. Jedes Bauelement der 74er Serie ist folgendermaßen bezeichnet:



PLD (Programmable Logic Device):

Programmierbare Logikelemente sind Strukturen, die softwaremäßig mit Logikfunktionen programmiert werden können. Die Entwicklung erfolgt in einer Logikprogrammiersprache (VHDL).

Beispiele solcher programmierbaren Logiken sind:

ALTERA „MAX7000" Serie

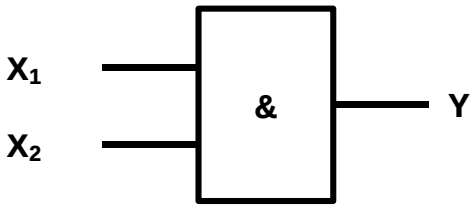
XILINX FPGA X4000 Serie

TEXAS INSTRUMENTS GAL Familie

Für Details siehe weiterführende Literatur.

V. Reversible Gatter:

Betrachtet man ein AND Gatter, so sieht man, daß sich aus dem logischen Ausgangszustand nicht mehr reproduzieren läßt, welcher Eingangszustand am Gatter angelegt wurde.



$$Y = X_1 \wedge X_2$$

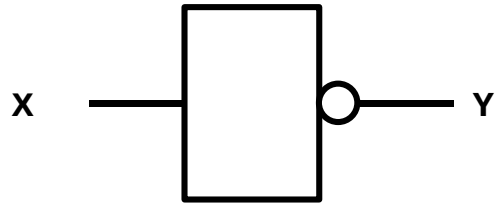
X_1	X_2	**Y**
0	0	0
0	1	0
1	0	0
1	1	1

Information geht verloren

So ist zwar klar daß bei einem logischen Ausgang 1 beide Eingänge logisch null waren, aber über die Eingangssituation bei einem Ausgang logisch null ist nichts bekannt. Daher ist das AND Gatter ein irreversibles Gatter.

Wie man sieht ist aber z. B der Inverter eindeutig reversibel. Zu jedem Ausgangszustand ist eindeutig definiert welcher Eingangszustand ihn erzeugt hat.

Inverter:

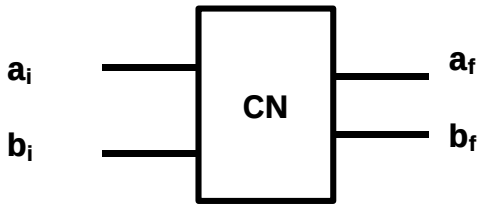


$$Y = \overline{X}$$

X	Y
0	1
1	0

Es sollen jetzt einige reversible Gatter vorgestellt werden, die für quantenmechanische Logiksysteme aufgrund Ihrer Reversibilität von Bedeutung sind.

Control-Not Gatter (CN-Gatter):

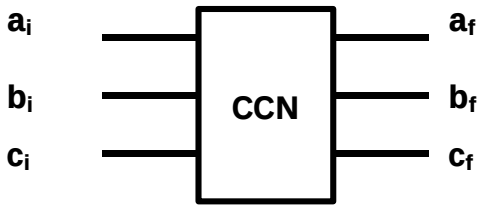


a_i	b_i	a_f	b_f
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Das CN Gatter verfügt über ein Control Bit a und ein Target bit b. Das control bit ändert seinen Wert während der Operation nicht. Das Target bit invertiert seinen logischen Zustand wenn das Control bit logisch 1 ist.
Da das Control bit unverändert bleibt ist auch der Zustand des Target bits vor der logischen Operation determiniert. Damit ist dieses Gatter reversibel.
Vergleicht man die Wahrheitstabelle mit den Grundelementen, so sieht man, daß die Logikfunktion für b_f durch ein XOR gegeben ist. Die boolschen Gleichungen lauten daher:

$$a_f = a_i$$
$$b_f = a_i \oplus b_i$$

Control-Control-Not Gatter (CCN-Gatter):



$$a_f = a_i$$
$$b_f = b_i$$
$$c_f = a_i b_i \oplus c_i$$

Repräsentation des AND bei $c_i = 0$

Repräsentation des Inverters bei $a_i = 1$ und $b_i = 1$

Repräsentation des CN Gatters bei $a_i = 1$

a_i	b_i	c_i	a_f	b_f	c_f
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Das 3 Bit CCN Gatter ist ein universelles reversibles Gatter, mit dem verschiedene Logikfunktionen realisiert werden können.

F-Gate (Fredkin Gate):

Das Fredkin Gatter wird auch Control-Exchange genannt. Seine Funktion ist charakterisiert durch die folgende Wahrheitstabelle:

Repräsentation des AND bei $c_i = 0$

a_i	b_i	c_i	a_f	b_f	c_f
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

Bei $a_i = 1$ tauschen die Bits b_i and c_i die Werte, daher auch der Begriff „Exchange Gate". Das F-Gate ist wie man sieht auch ein reversibles Gatter.

VI. Weiterführende Literatur

[1]	Tietze, Schenk	„Halbleiter-Schaltungstechnik“ Springer Verlag ; ISBN 3-540-56184-6
[2]	Weste, Eshraghian	„Principles of CMOS VLSI Design“ Addison Wesley ; ISBN 0-201-53376-6
[3]	S. Nowick	„Automatic Synthesis of Burst Mode asynchronous controllers“ Technical report CSL-TR-95-686 Stanford University
[4]	M.W. Shields	„Serielle und parallele Automaten“ VCH Verlagsgesellschaft ; 1987