

1 Notation

$x \rightarrow y := \neg x + y$

AND	\wedge	\cdot
OR	\vee	$+$
NOT	\neg	$\overline{}$
NAND	\uparrow	$\overline{A \cdot B}$
NOR	\downarrow	$\overline{A + B}$
XOR	\oplus	\neq
XNOR	\odot	\equiv

Rechengesetze

Kommutativgesetze

$A \cdot B = B \cdot A$
 $A + B = B + A$

Assoziativgesetze

$(A \cdot B) \cdot C = A \cdot (B \cdot C)$
 $(A + B) + C = A + (B + C)$

Distributivgesetze

$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
 $A + (B \cdot C) = (A + B) \cdot (A + C)$

Identitätsgesetze

$A \cdot 1 = A$
 $A + 0 = A$

Negationsgesetze

$A \cdot \neg A = 0$
 $A + \neg A = 1$

Idempotenzgesetze

$A \cdot A = A$
 $A + A = A$

Null- und Einselementgesetze

$A \cdot 0 = 0$
 $A + 1 = 1$

Absorptionsgesetze

$A \cdot (A + B) = A$
 $A + (A \cdot B) = A$

De Morgan'sche Gesetze

$\neg(A \cdot B) = \neg A + \neg B$
 $\neg(A + B) = \neg A \cdot \neg B$

2 Beweise

$\neg 1 = 0$

$\neg 1 = \neg 1 + 0$ (Neutrales Element 0)
 $= \neg 1 + \neg \neg 0$ (Idempotenz)
 $= \neg(1 \cdot \neg 0)$ (De Morgan)
 $= \neg \neg 0$ (Neutrales Element 1)
 $= 0$ (Idempotenz)

Keine 3-elementige Boolesche Algebra

Angenommen, es existiert eine 3-elementrige Boolesche Algebra $M = \{1, 0, a\}$. Für $\neg a$ gibt es 3 jeweils widersprüchliche Möglichkeiten:

$\neg a = 0 \rightarrow 1 = a + \neg a = a + 0 = a$
 $\neg a = 1 \rightarrow 0 = \neg a \cdot a = 1 \cdot a = a$
 $\neg a = a \rightarrow 1 = a + \neg a = a + a = a$

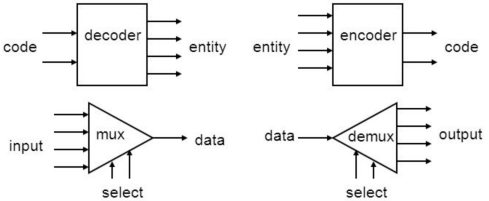
Funktionale Vollständigkeit von $\{\rightarrow, \oplus\}$

Es ist zu zeigen, dass die Menge $\{\rightarrow, \oplus\}$ funktional vollständig ist, gegeben dass $\{+, \neg\}$ funktional vollständig ist.

Da jede Funktion mit $+$ und \neg darstellbar ist, genügt es zu zeigen, dass sowohl $+$ als auch \neg mit \rightarrow und \oplus ausgedrückt werden können:

$\neg x = 1 \oplus x$
 $= (x \rightarrow x) \oplus x$
 $x + y = \neg x \rightarrow y$
 $= ((x \rightarrow x) \oplus x) \rightarrow y$

Digitale Logikbausteine



Multiplexer (Mux)

Ein Multiplexer (Mux) wählt eines von vielen Eingangssignalen aus und leitet dieses auf einen einzelnen Ausgang. Er fungiert als mehrere-zu-eins-Schalter.

n Daten $+$ $\log_2(n)$ Steuerung

- Auswahl eines Datenkanals aus mehreren Quellen.
- Routing eines Signals basierend auf einer Steueradresse.

Demultiplexer (Demux)

Ein Demultiplexer (Demux) nimmt ein einzelnes Eingangssignal und leitet es auf eines von vielen Ausgangssignalen. Er funktioniert als eins-zu-mehrere-Schalter.

$\log_2(n)$ Steuerung $+$ n Ausgänge

- Verteilung eines Datenkanals auf mehrere Ausgänge.
- Erzeugung mehrerer Steuersignale aus einem einzigen Eingang.

Decoder

Ein Decoder hat n binäre Eingangsleitungen und 2^n Ausgangsleitungen. Für jede Eingangskombination wird genau einer der Ausgänge aktiviert.

n Eingänge $\rightarrow 2^n$ Ausgänge

- Jede Eingangskombination aktiviert genau einen Ausgang.
- Einsatz z.B. in digitalen Anzeigesystemen, um eine binäre Zahl in eine spezifische Anzeige zu übersetzen.

Encoder

Ein Encoder hat 2^n Eingangsleitungen, von denen jeweils nur eine aktiv sein darf, und wandelt diese in eine n -Bit binäre Zahl um.

2^n Eingänge $\rightarrow n$ Ausgänge

- Wandelt den aktiven Eingang in eine binäre Darstellung um.
- Oft in Tastaturen genutzt, um die gedrückte Taste in einen Binärcode umzuwandeln.

Schaltfunktionen

2-De-Mux (1-to-4-Demultiplexer)

$z_0(x, y_0, y_1) = x(\neg y_0 \neg y_1)$
 $z_1(x, y_0, y_1) = x(\neg y_0 y_1)$
 $z_2(x, y_0, y_1) = x(y_0 \neg y_1)$
 $z_3(x, y_0, y_1) = x(y_0 y_1)$

Darstellung von Funktionen mittels Multiplexer

Eine Funktion $f : B^n \rightarrow B$ kann durch einen $(n-1)$ -Multiplexer dargestellt werden, indem die ersten $(n-1)$ Variablen als Steuersignale und die n -te Variable als Eingangssignal verwendet werden. Abhängig von den Steuersignalen wird der Mux-Ausgang entweder x_n , $\neg x_n$, 0 oder 1 sein.

Beispiel für $B^3 \rightarrow B$ mit einem 2-Multiplexer:

x_0	x_1	x_2	z	
0	0	0	0	$\rightarrow 0$
0	0	1	0	
0	1	0	0	
0	1	1	1	$\rightarrow x_2$
1	0	0	1	$\rightarrow 1$
1	0	1	1	
1	1	0	1	$\rightarrow \neg x_2$
1	1	1	0	

Basisumwandlung

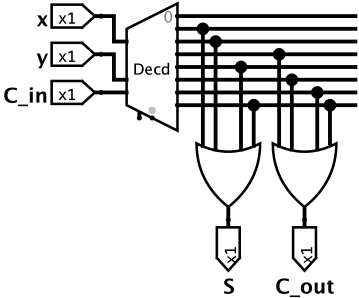
$(132)_{10} \rightarrow (204)_8$

$132 \div 8 = 16 \text{ Rest } 4$
 $16 \div 8 = 2 \text{ Rest } 0$
 $2 \div 8 = 0 \text{ Rest } 2$

3 Beispielaufgaben

3.1 Volladdierer mit 3:8 Decoder

A	B	C_{in}	S	C_{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



3.2 Quine McCluskey Verfahren

$f(a, b, c, d) = \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$

	Column I	Column II	Column III
group 0	0 0000 ✓ 1 0001 ✓	0, 1 000- ✓ 0, 2 00-0 ✓	0, 1, 8, 9 -0-0 0, 2, 8, 10 -0-0
group 1	2 0010 ✓ 8 1000 ✓	0, 8 -000 ✓ 1, 5 0-01 ✓	0, 8, 1, 9 -0-0 0, 8, 2, 10 -0-0
group 2	5 0101 ✓ 6 0110 ✓ 9 1001 ✓ 10 1010 ✓	1, 9 -001 ✓ 2, 6 0-10 ✓ 2, 10 -010 ✓ 8, 9 100- ✓ 8, 10 10-0 ✓	2, 6, 10, 14 --10 2, 10, 6, 14 --10
group 3	7 0111 ✓ 14 1110 ✓	5, 7 01-1 ✓ 6, 7 011- ✓ 6, 14 -110 ✓ 10, 14 1-10 ✓	

Nicht weiter verwendete Minterme:

$f = \overline{a}cd + \overline{a}b\overline{d} + \overline{a}bc + \overline{b}\overline{c} + \overline{b}d + c\overline{d}$

Term	0	1	2	5	6	7	8	9	10	14
$\overline{a}cd$		1		1						
$\overline{a}b\overline{d}$				1		1				
$\overline{a}bc$					1	1				
$\overline{b}\overline{c}$	1	1					1	1		
$\overline{b}d$	1		1				1		1	
$c\overline{d}$			1		1			1	1	1

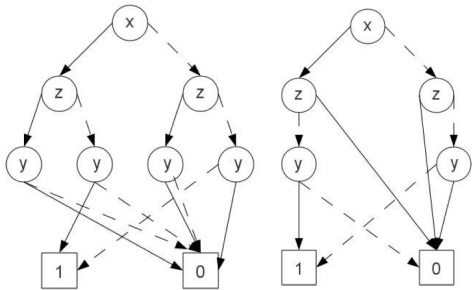
Minimale Darstellung:

$f = \overline{a}bd + \overline{b}\overline{c} + c\overline{d}$

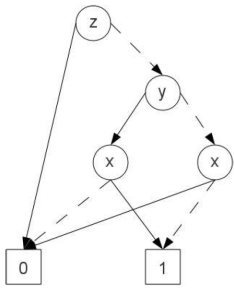
3.3 OBDD

x	y	z	$f(x, y, z)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

OBDD mit $x > z > y$



Optimiertes OBDD mit $z > y > x$



3.4 Unabhängige Fehlerdiagnose

$$f(x_0, x_1, x_2) = x_0(x_1 + x_2)$$

Wertetabelle:

x_0	x_1	x_2	$f(x_0, x_1, x_2)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Testpaare:

- x_0 : {(000), (100)}, {(010), (110)}, {(011), (111)}
- x_1 : {(101), (111)}
- x_2 : {(100), (101)}

Minimale Testmengen:

- {(000), (100), (101), (111)}
- {(011), (100), (101), (111)}

3.5 Eliminieren von Schaltungshazards

Anwenden von Satz von Eichelberger. Alle Primimplikanten in DNF führt zu Hazard-freiem Verhalten.

3.6 Schaltungsabhängige Fehlerdiagnose

$$f(x_0, x_1) = x_0 \oplus x_1$$

Annahme: $f_{x_0} = x_0$ Stuck zero
 $f_{x_1} = x_1$ Stuck zero

x_0	x_1	f	f_{x_0}	f_{x_1}
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	0	1	1

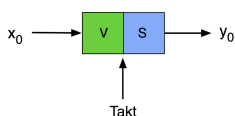
x_0	x_1	f	$f \oplus f_{x_0}$	$f \oplus f_{x_1}$
0	0	0	0	0
0	1	1	0	1
1	0	1	1	0
1	1	0	1	1

Minimale Testmenge:

- {(1, 1)}

Alle Fehler sind feststellbar.

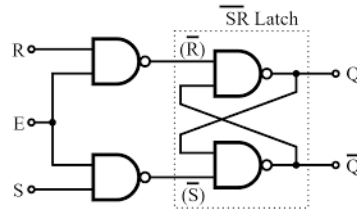
4 Delays



- Synchronisiertes Delay durch zentrale Uhr
- Taktimpulse steuern Rückkopplungs-Delay
- Sperrschaltung zwischen Vor- und Hauptspeicher
- Arbeitsphase: Ausgabe des Hauptspeichers (S)
- Setzphase: Übertrag von Vor- (V) zu Hauptspeicher (S)
- Kurzzeitiges Öffnen der Sperre während Setzphase

4.1 Flipflops

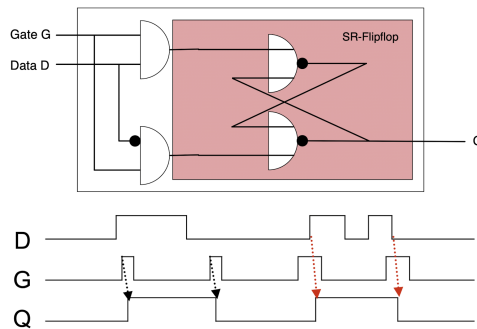
4.1.1 SR-Flipflop



- Zwei Eingänge: Set (S), Reset (R)
 - Flanken- oder pegelgesteuert konfigurierbar
 - Flankensteuerung: Zusätzliche Logik (z.B. Taktflankendetektor)
 - Vermeiden von gleichzeitigem S=R=1
 - Race Condition, falls S und R fast gleichzeitig von 0 auf 1 wechseln
 - Master-Slave für definierte Ausgabe bei Flanke
- Anlegen eines **Taktsignals** erlaubt bessere Kontrolle.

S	R	$Q(t_0)$	$Q(t_1)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	verboten	

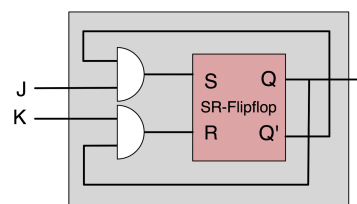
4.1.2 D-Flipflop



- Ein Daten-Eingang (D), ein Takt-Eingang (Clock)
- Flankengetriggert: Übernahme von D bei Taktflanke
- Keine verbotenen Zustände
- Speichert einzelnes Bit
- Auch in Master Slave Schaltung möglich (Edge-Triggered)

D	G	$Q(t_0)$	$Q(t_1)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

4.1.3 JK-Flipflop

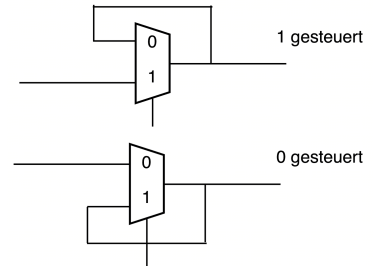


- Zwei Eingänge: J (Set), K (Reset)
- Flankengetriggert: Toggeln bei J=K=1
- Interne Rückkopplung verhindert undefinierte Zustände
- Master-Slave-Variante für sequenzielle Logik

- Geeignet für Zähler und Register
- Setzen nur möglich wenn $Q = 0$, Reset nur möglich wenn $Q = 1$

J	K	S	R	$Q(t_0)$	$Q(t_1)$
0	0	0	0	0	0
0	0	0	0	1	1
0	1	0	0	0	0
0	1	0	1	1	0
1	0	1	0	0	1
1	0	0	0	1	1
1	1	1	0	0	1
1	1	0	1	1	0

4.2 MUX-Delay



- MUX Delay: Zeitverzögerung beim Durchschalten der Daten.
- MUX Konfiguration für einfache Flanke: Takt direkt an Steuereingang.
- MUX Konfiguration für doppelte Flanke: Takt und invertierter Takt auf Steuereingänge zweier MUXe, die abwechselnd aktiviert werden.

5 Schaltwerke

Einer- und Zweierkomplement

Einerkomplement:

- Invertierung jedes Bits (0 wird zu 1, 1 wird zu 0).
- Beispiel: Einerkomplement von 0101 ist 1010.
- Zwei Darstellungen für Null (0000 und 1111).
- "Overflow" wird an der niedrigsten Stelle adiert.

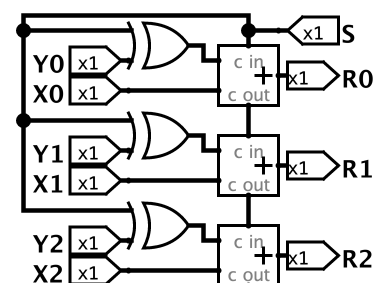
$$\begin{aligned} \text{Beispiel: } & -63_{(10)} - 27_{(10)} \\ & -63_{(10)} \rightarrow -(0011\ 1111)_{(2)} \rightarrow 1100\ 0000_{(2)} \\ & -27_{(10)} \rightarrow -(0001\ 1011)_{(2)} \rightarrow 1110\ 0100_{(2)} \\ & \begin{array}{r} 1100\ 0000_{(2)} \\ +1110\ 0100_{(2)} \\ \hline 11010\ 0100_{(2)} \end{array} \rightarrow 1010\ 0101_{(2)} \\ & 1010\ 0101_{(2)} \rightarrow -(0101\ 1010)_{(2)} \rightarrow -90_{(10)} \end{aligned}$$

Zweierkomplement:

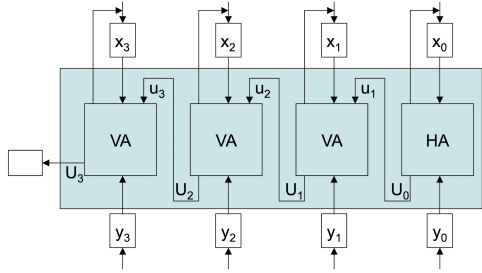
- Addition von 1 zum Einerkomplement.
- Beispiel: Zweierkomplement von 0101 ist 1011.
- Eine Darstellung für Null, vereinfacht binäre Subtraktion.

! Aufpassen beim zurückwandeln, auch hier wieder +1

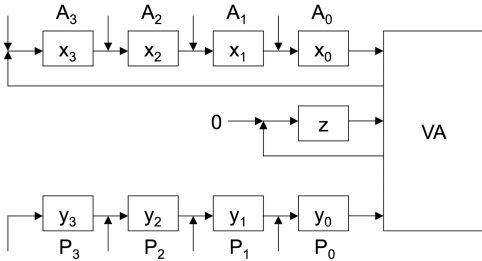
3-Bit Addier-/Subtrahierwerk



4-Bit Paralleladdierwerk



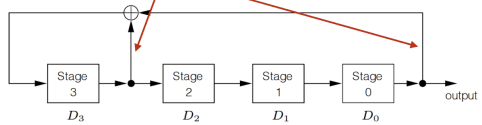
4-Bit Serienaddierwerk



Ein n -Bit Serienaddierer benötigt $n + 1$ Schritte. 1 Schritt für das Laden der Register und dann n Addierschritte.

Linear rückgekoppeltes Schieberegister

Das LRSR ($4, 1 + X + X^4$) hat folgende Schaltung



Keystream kann durch "rückwärts Einfädeln" überprüft werden.

6 Gleitkommadarstellung

$$z = \pm m \cdot b^{\pm e}$$

m : Mantisse; b : Basis; e : Exponent

Nachkommastellen umrechnen

$$\begin{aligned} 0.703125_{(10)} \times 2 &= 1.40625_{(10)} \\ 0.40625_{(10)} \times 2 &= 0.8125_{(10)} \\ 0.8125_{(10)} \times 2 &= 1.625_{(10)} \\ 0.625_{(10)} \times 2 &= 1.25_{(10)} \\ 0.25_{(10)} \times 2 &= 0.5_{(10)} \\ 0.5_{(10)} \times 2 &= 1.0_{(10)} \\ 0.703125_{(10)} &\rightarrow 0.1011010_{(2)} \end{aligned}$$

Normalisierung

Definition 1

$$\frac{1}{b} \leq |m| < 1$$

Definition IEEE 754: Eine Gleitkommazahl $\pm m \cdot 2^{\pm e}$ heisst normalisiert, falls $1 \leq m < 2$
Form: $(-1)^V \cdot 1.M \cdot 2^{E-\text{Basis}}$ wo (Basis = 127; $E = e + \text{Basis}$)
1 Bit V + 8 Bit E + 23 Bit M

E	M	Bedeutung
= 255	$\neq 0$	NAN / ungültig
= 255	= 0	$\pm \infty$
$0 < E < 255$		$(-1)^V \cdot (1.M) \cdot 2^{E-127}$
= 0	$\neq 0$	$(-1)^V \cdot (0.M) \cdot 2^{-126}$
= 0	= 0	± 0

7 ASCII

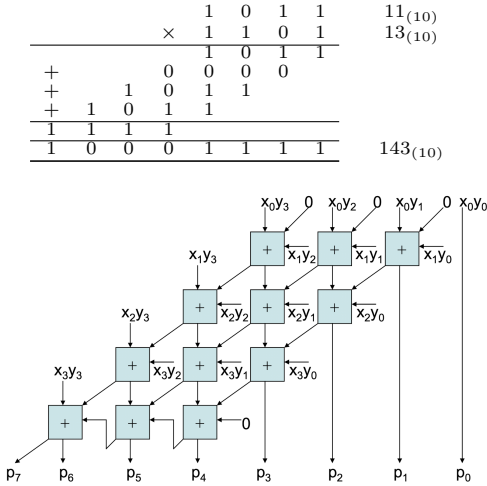
P000 0000

- $P = 0$: Anzahl von „1“-Bits ist gerade
- $P = 1$: Anzahl von „1“-Bits ist ungerade

Base 10	Range
0 - 31	Control characters
32	Space
33 - 47	! " # \$ % & ' () * + , - . /
48 - 57	0 1 2 3 4 5 6 7 8 9
58 - 64	: ; < = > ? @
65 - 90	A - Z
91 - 96	[\] ^ _
97 - 122	a - z
123 - 126	{ } ~
127	DEL

8 Multiplikation

Carry-Save-Multilikation, gleich wie schriftliches Multiplizieren.



9 Division

Wichtig: Zuerst checken dass $2^4 d > z_{[5-8]}$ ansonsten Overflow

Wichtig: Zweierkomplement bilden für $-2^4 d$

Restoring Division

z	0 1 1 1	0 1 0 1 = $117_{(10)}$
$2^4 d$	0 1 0 1 0	= $10_{(10)}$
$-2^4 d$	1 0 1 1 0	
$s^{(0)}$	0 0 1 1 1	0 1 0 1
$2s^{(0)}$	1 1 1 1 0	1 0 1
$+(-2^4 d)$	1 0 1 1 0	
$s^{(1)}$	0 0 1 0 0	1 0 1 $q_3 = 1$
$2s^{(1)}$	0 1 0 0 1	0 1
$+(-2^4 d)$	1 0 1 1 0	
$s^{(2)}$	1 1 1 1 1	0 1 $q_2 = 0$
$s^{(2)} = 2s^{(1)}$	0 1 0 1 1	0 1
$2s^{(2)}$	1 1 0 1 0	1
$+(-2^4 d)$	1 0 1 1 0	
$s^{(3)}$	0 1 0 0 0	1 $q_1 = 1$
$2s^{(3)}$	1 0 0 0 1	
$+(-2^4 d)$	1 0 1 1 0	
$s^{(4)}$	0 0 1 1 1	$q_0 = 1$
s		0 1 1 1 = $7_{(10)}$
q		1 0 1 1 = $11_{(10)}$

Non-restoring Division

z	0 1 1 0	1 0 0 1 = $100_{(10)}$
$2^4 d$	0 1 0 0 1	= $9_{(10)}$
$-2^4 d$	1 0 1 1 1	
$s^{(0)}$	0 0 1 1 0	0 1 0 0
$2s^{(0)}$	0 1 1 0 0	1 0 0
$+(-2^4 d)$	1 0 1 1 1	
$s^{(1)}$	0 0 0 1 1	0 0 $q_3 = 1$
$2s^{(1)}$	0 0 1 1 1	0 0
$+(-2^4 d)$	1 0 1 1 1	
$s^{(2)}$	1 1 1 1 0	0 0 $q_2 = 0$
$2s^{(2)}$	1 1 1 0 0	0
$+2^4 d$	0 1 0 0 1	
$s^{(3)}$	0 0 1 0 1	0 $q_1 = 1$
$2s^{(3)}$	0 1 0 1 0	
$+(-2^4 d)$	1 0 1 1 1	
$s^{(4)}$	0 0 0 0 1	$q_0 = 1$
s		0 0 0 1 = $1_{(10)}$
q		1 0 1 1 = $11_{(10)}$

10 Model checking

Erreichbarkeitsanalyse

Gegeben seien die Inputmenge $I = \{0, 1\}$, die Zustandsmenge $Q = \{0, 1\}^4$ und die Übergangsfunktion $d: I \times Q \rightarrow Q$ mit

$$d(0; q_1, q_2, q_3, q_4) = \begin{cases} (q_1 q_2 q_3 q_4)_2 + (100)_2 & \text{if } < 16 \\ (0, 0, 0, 0) & \text{else} \end{cases}$$

$$d(1; q_1, q_2, q_3, q_4) = \begin{cases} (q_1 q_2 q_3 q_4)_2 + (101)_2 & \text{if } < 16 \\ (0, 0, 0, 0) & \text{else} \end{cases}$$

Start: = (0, 1, 0, 0)

$S_0 = \{(0, 1, 0, 0)\}$

$d(0; 0, 1, 0, 0) = (1, 0, 0, 0)$

$d(1; 0, 1, 0, 0) = (1, 0, 0, 1)$

$S_1 = S_0 \cup \{(1, 0, 0, 0), (1, 0, 0, 1)\}$

$d(0; 1, 0, 0, 0) = (1, 1, 0, 0)$

$d(0; 1, 0, 0, 1) = (1, 1, 0, 1)$

$d(1; 1, 0, 0, 0) = (1, 1, 0, 1)$

$d(1; 1, 0, 0, 1) = (1, 1, 1, 0)$

$S_2 = S_1 \cup \{(1, 1, 0, 0), (1, 1, 0, 1), (1, 1, 1, 0)\}$

\vdots

$S_{n+1} = S_n \rightarrow \text{terminieren}$

$$S = \{(0, 0, 0, 0), (0, 1, 0, 0), (1, 0, 1, 0), (1, 0, 0, 0), (1, 0, 0, 1), (1, 0, 1, 0), (1, 1, 0, 0), (1, 1, 0, 1), (1, 1, 1, 0), (1, 1, 1, 1)\}$$

 $(1, 0, 1, 1) \notin S$

11 CTL

- $EX\phi$: es gibt einen Pfad, auf dem als nächstes ϕ wahr ist
- $AX\phi$: auf jedem Pfad ist als nächstes ϕ wahr
- $EF\phi$: es gibt einen Pfad auf dem irgendwann ϕ wahr ist
- $AF\phi$: auf allen Pfaden ist ϕ irgendwann wahr
- $EG\phi$: es gibt einen Pfad, auf dem ϕ immer wahr ist
- $AG\phi$: auf allen Pfaden ist ϕ immer wahr
- $\phi EU\psi$: es gibt einen Pfad, auf dem ϕ wahr bleibt, bis ψ wahr wird
- $\phi AU\psi$: auf allen Pfaden bleibt ϕ wahr, bis ψ wahr wird

Symbol	Bedeutung
\top	Wahr (True)
\perp	Falsch (False)
\neg	Negation (Not)
\wedge	Konjunktion (And)
\vee	Disjunktion (Or)
\rightarrow	Implikation
\leftrightarrow	Äquivalenz (If and only if)
p	Eine atomare Proposition
$L(v)$	Labeling-Funktion
I	Anfangszustände (Initial states)
π	Ein Pfad
π_i	Die i -te Stelle in einem Pfad
$\mathcal{P}(Prop)$	Potenzmenge d. atom. Propositionen
V	Knoten (States)
E	Kanten (Transitions)

- Auf jedem Pfad ist es immer möglich, dass p irgendwann wahr wird:

$\rightarrow AG\ EF\ p$

- p ist möglich:

$\rightarrow EF\ p$

- p und q sind nie gleichzeitig wahr:

$\rightarrow AG\neg(p \wedge q)$ oder $\neg EF(p \wedge q)$

- Auf jedem Pfad ist p unendlich oft wahr:

$\rightarrow AG\ AF\ p$

- p und immer falls p dann direkt danach p , impliziert p gilt immer (Induktion):

$\rightarrow p \wedge AG(p \rightarrow AXp) \rightarrow AGp$

- Auf jedem Pfad sind p und q abwechselnd wahr, wobei jeweils nur eine der beiden Propositionen wahr ist:

$\rightarrow (p \wedge \neg q) \vee (\neg p \wedge q) \wedge AG((p \rightarrow AX(\neg p \wedge q)) \wedge (q \rightarrow AX(p \wedge \neg q)))$

- Sei $p =$ 'Es regnet'. Es regnet und es wird weiterregnen, bis es aufgehört hat, zu regnen:
- $\rightarrow p \wedge (p\ AU\ \neg p)$

Algorithmus

Reduktionen

- $AX\varphi \Leftrightarrow \neg EX\neg\varphi$
 - Wenn $AX\varphi$ wahr ist, dann gibt es keinen Pfad, auf dem als nächstes $\neg\varphi$ wahr ist.
- $AG\varphi \Leftrightarrow \neg EF\neg\varphi$
 - Wenn $AG\varphi$ wahr ist, dann gibt es keinen Pfad, auf dem irgendwann $\neg\varphi$ wahr ist.
- $EF\varphi \Leftrightarrow TEU\varphi$
 - Wenn $EF\varphi$ wahr ist, dann ist von einem wahren Zustand aus φ irgendwann erreichbar.
- $EG\varphi \Leftrightarrow \neg AF\neg\varphi$
 - Wenn $EG\varphi$ wahr ist, dann ist φ immer wahr auf einem Pfad, auf dem $\neg\varphi$ nicht irgendwann wahr wird.
- $\varphi AU\psi \Leftrightarrow \neg((\neg\psi)EU(\neg(\varphi \vee \psi))) \wedge AF\psi$
 - Wenn $\varphi AU\psi$ wahr ist, dann gibt es keinen Pfad, auf dem $\neg\psi$ bleibt, bis $\neg(\varphi \vee \psi)$ wahr wird, und irgendwann wird ψ wahr.

Hierbei ist T definiert als $T := p_0 \vee \neg p_0$, was immer wahr ist.

Der Algorithmus behandelt:

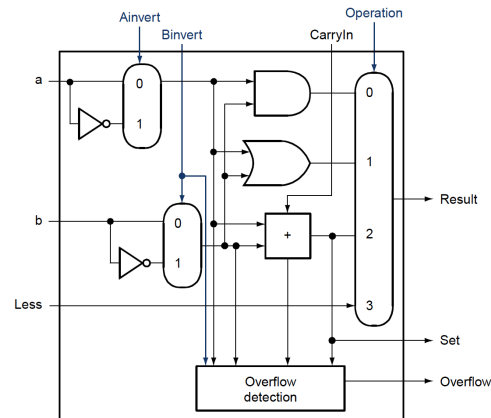
\neg, \wedge, EX, EU, AF

Es gilt:

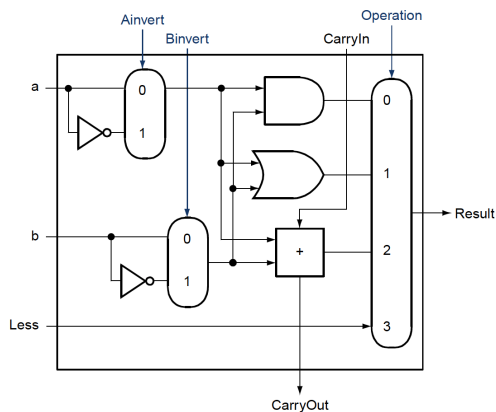
$$\begin{aligned}
 AX\Phi &\equiv \neg EX\neg\Phi \\
 AG\Phi &\equiv \neg EF\neg\Phi \\
 EF\Phi &\equiv TEU\Phi \\
 EG\Phi &\equiv \neg AF\neg\Phi \\
 \Phi AU\Psi &\equiv \neg(\neg\Psi EU\neg(\Phi \vee \Psi)) \wedge AF\Psi \\
 \Phi \rightarrow \Psi &\equiv \neg\Phi \vee \Psi \\
 \Phi \vee \Psi &\equiv \neg(\neg\Phi \wedge \neg\Psi)
 \end{aligned}$$

12 ALU

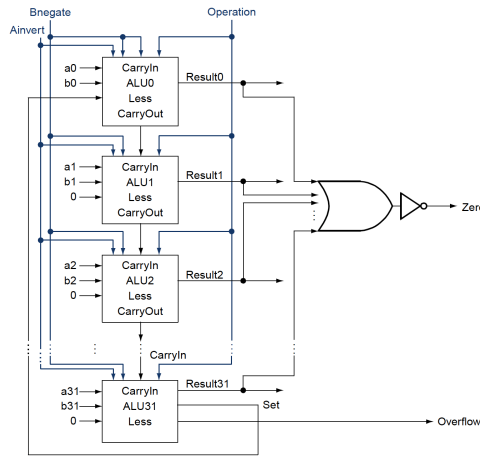
Most-significant-Bit



Every other Bit



Komplettes ALU (bemerke $\neg B$ als c_{in} im least-significant-bit für 2-er Komplement)



$$slt(a, b) := \begin{cases} 0 & \text{sonst} \\ 1 & \text{wenn } a < b \end{cases}$$