EIR v2.1

EIR Cheatsheet. V2 - Braucht weniger platzt. MIT, https://github.com/eisenwinter/fh-hgb-stuff
Jan Caspar, Aktualisiert 30. Jänner 2017

Zweierkomplement

$X_{10} \overset{Konvertierung}{\rightarrow} X_{2k}$

1. Zehner ins zweier System umrechnen, zB $-5_{10} = -0101_2$
2. Alle „Bits" umdrehen, zB -1010_2
3. Plus 1 rechnen, zB 1011_{2k}

Sign-Extension $1011_{2k} = 11......11011_{2k}$ und $0011_{2k} = 0000....0011_{2k}$.

Achtung beim **Addieren** stets Bits auffüllen mittels Sign-Extension.

Overflow Achtung auf Übertrag bzw. ob vernachlässigbar.

Schnellkonvertierung Ersten 1er finden, danach Bitflip. Bsp: $0010|\overset{\downarrow}{1}0 \rightarrow 110110$

Schaltalgebra

Rechenregeln

$\wedge = *, \vee = +,$

Vereinfachen Reihenfolge: Distributivität, Komplementär, Neutralität, Distributivität Inverse

DeMorgan $\overline{x+z} \equiv \overline{x} * \overline{y} \leftrightarrows \overline{x*y} \equiv \overline{x} + \overline{y}$

Distributivität $x * (y+z) \equiv x*y + x*z \leftrightarrows x + (y*z) \equiv (x+y)*(x+z)$

Komplementär $x + \overline{x} \equiv 1 \leftrightarrows x * \overline{x} \equiv 0$

Neutralität $x + 0 \equiv x \leftrightarrows x * 1 \equiv x$

Idempotenz $x + x \equiv x \leftrightarrows x * x \equiv x$

Extremalität $x + 1 \equiv 1 \leftrightarrows x * 0 \equiv 0$

Absorption $x * y + x \equiv x \leftrightarrows (x+y) * x \equiv x$

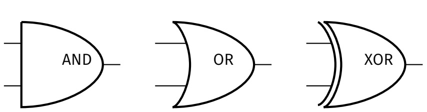
Involution $\overline{\overline{x}} \equiv x$

Assoziativität $(x+y)+z \equiv x + (y+z) \leftrightarrows (x*y)*z \equiv x*(y*z)$

DNF & KNF

KNF	Off-Menge	0er aus Wertetabelle
$(a \vee b \vee c) \wedge \ldots$		Werte negieren
DNF	On-Menge	1 aus Wertetabelle
$(\neg a \wedge \neg b \wedge \neg c) \vee \ldots$		Werte gleich

Schematic



KV-Diagramm

1. Diagramm erstellen
2. Primimplikanten suchen
3. wesentliche Primimplikanten suchen
4. minimale Überdeckung suchen

Ein Primimplikant heißt **Kern-Primimplikant**, wenn es einen Minterm gibt, der nur diesen Primimplikanten impliziert. Ein Primimplikant heißt **wesentlicher** Primimplikant, wenn er in jeder minimalen Form einer Schaltfunktion enthalten ist und kein Kernprimimplikant ist.

x	y	z	v	f
0	0	0	0	0
...				

	$\overline{zv}$	$\overline{z}v$	zv	$z\overline{v}$
$\overline{xy}$	0	0	0	0
$\overline{x}y$	0	1	1	0
xy	0	1	1	0
$x\overline{y}$	1	1	0	0

KKNF $(x+y) * (w + \overline{y} + v) * (y + \overline{z})$

DKNF $vy + \overline{y}x\overline{z}$

u	v	w	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

	vw	$v\overline{w}$	$\overline{vw}$	$\overline{v}w$
u	1	0	1	1
$\overline{u}$	0	0	1	0

KKNF $(v+w) * (\overline{u} + w)$

Umsetzung minimal / auschließlich NAND oder NOR Gatter

NOR KV-Diagram, KKNF bilden, DeMorgan anwenden. Beispiel

$y = (v + \overline{w}) * (\overline{u} + w) \longrightarrow \overline{\overline{(v + \overline{w}) * (\overline{u} + w)}} \equiv \overline{\overline{(v + \overline{w})} + \overline{(\overline{u} + w)}}$

NAND KV-Diagram, DKNF bilden, DeMorgan anwenden. Beispiel

$y = \overline{B}\,\overline{C} + BC + AB \longrightarrow \overline{\overline{\overline{B}\,\overline{C}} + \overline{BC} + \overline{AB}}$

Zeichen & Kodierungen

Hammingcode

Alle Bits an der Position von zweier Potenzen (1, 2, 4, 8, 16, 32, 64, etc.) sind Paritätsbits, aller anderen sind die Datenbits. Jeder Paritätsbit berechnet die Parität von einigen der Bits. Position 1: Checkt 1 Bit, Springt 1 Bit, Checkt 1 Bit, Springt 1 Bit, etc. (1,3,5,7,9,11,13,15,...), Position 2: Checkt 2 Bits, Sprint 2 Bits, Checkt 2 Bits, etc. (2,3,6,7,10,11,14,15,...), usw für alle Paritätsbits. Bei einer ungerade Anzahl von Paritätsbits 1, bei einer gerade Anzahl 0. Um Fehler zu finden sind die Paritätsbits zu Prüfen. Ein Fehler lässt sich korrigieren, sollten zum Beispiel die Bits 2 und 8 falsch sein, so ist der Fehler auf 2+8= Position 10.

Hammingabstand

Unterschiede in Bitfolge zB

11**100**110
11**011**110
Abstand: 3

Graycode

- x_1 =Dualzahl
- x_2 =Rechtsshift von x_1 um 1 Bit
- $x_3 = x_1 \oplus x_2$, also x_1XOR $x_2$ist das die gewünschte Zahl im Gray-Code.

Automaten

λ Ausgabefunktion

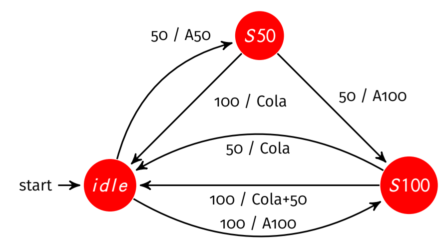
δ Zustandsüberführungsfunktion

Moore Automat

Beispiel Cola Automat, Preis 1,50, akzeptiert 1 €, 0,50 €

I $= \{100, 50\}$

O $= \{A50, A100, Cola, (Cola + 50)\}$



δ	50	100
idle	S50	S100
S50	S100	idle
S100	idle	idle

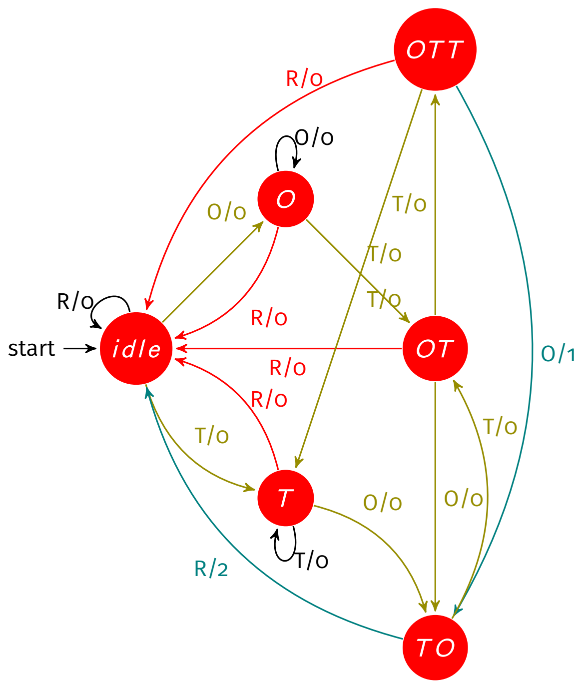
λ	50	100
idle	A50	A100
S50	A100	Cola
S100	Cola	Cola

Mealy Automat („Antwort“)

Beispiel OTTO (Ausgabe 1), TOR (Ausgabe 2) überlappend erkennen

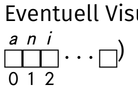
$\mathbf{I} = \{O, T, R\}$

$\mathbf{O} = \{0, 1, 2\}$



Zustandmenge $z = \{idle, O, T, OT, TO, OTT\}$

μBefehle

1. Speicherplätze definieren (Bsp. $R_0 \ldots a + n, R_2 \ldots index, etc$)
2. Eventuell Visualisierung des Speichers aufzeichnen (Bsp. 

)
3. Flowchart des Ablaufs erstellen
4. Sprungvorbereitungen (SVB) suchen
5. Sprungziele markieren
6. Befehlesschritt festlegen (Nummeriern) und Segmente einzeichnen

Merke Lese- und Schreibzugriffe, Sprungvorbereitung erfordern **2 Befehle.** Ein Segment besteht auf 4 Befehlen, und ein Sprung funktioniert nur in Segmentsprüngen.

Lese-/Schreibzugriff Adresse einstellen, Wert lesen/schreiben (2 Befehle)

Achtung wenn Speicher wartet, warten 1 Byte (w1b), 1 Byte weil Argument notwendig

Stop Ist immer ein Sprungziel

Maske ZPNOf (Zero, Positive, Negative, Overflow)

Conditions Größervergleich: X SUB Y und Positiv-Flag (Maske 0100) prüfen, für gleich X SUB Y und Zero-Flag (Maske 1000) prüfen.

Adressindex Sollte in einem Register ein Index für eine Speicheradresse stehen so muss man diese vorher durch die ALU schleifen (damit Sie ins MAR kommt). Das heißt alle Schritte wo ein Wert von einem Index gelesen werden muss erfordern zwei Befele.

Sprungmaske

S	M	S	M	
0	0			CAR = 4xCN (freier Sprung)
0	1			CAR=CAR+1+4*CN
1	0			CAR=CAR-1+4*CN
1	1	0	0	CAR = 4*COP (bed. Sprung)
1	1	0	1	Maske + Flags <> 0 CAR=4*COP
1	1	1	0	Maske + Flags = 0 CAR=4*COP
1	1	1	1	CAR = CAR++

μBefehle-Beispiele

Lesen (aus RAM) ([0] $\rightarrow$ R0)

0	1	H	-
		R	$Z := \mathbf{0}, CAR++$
		B	$z \rightarrow MAR$
	Lesen, 1 Byte		
	2	H	$[MAR] \rightarrow MDR, \boldsymbol{R_0} \rightarrow Y$
		R	$Z := Y, CAR++$
		B	$Z \rightarrow R_0$

Schreiben (auf RAM) (R2 $\rightarrow$ [6])

0	1	H	-
		R	$Z := \mathbf{6}, CAR++$
		B	$z \rightarrow MAR$
	Warten, 1 Byte		
	2	H	$\boldsymbol{R_2} \rightarrow X$
		R	$Z := X, CAR++$
		B	$Z \rightarrow MDR \rightarrow [MAR]$
Schreiben, 1 Byte			

SVB (Sprung auf Seg. 4)

0	1	H	-
		R	$Z := \mathbf{4}, CAR++$
		B	$z \rightarrow MDR$
	warten, 1 Byte		
	2	H	-
		R	-
		B	$MDR \rightarrow COP$
	warten, 1 Byte		

Sprung (wenn $R_1 = R_2$), wenn nein CAR++, sonst 4*COP

0	1	H	$R_1 \rightarrow x, R_2 \rightarrow y$,w1b
		R	$Z := X\ SUB\ Y, CAR = 4 * COP$ Maske = 1000 (<>0)
		B	

Zahl von RAM ([2]=5) lesen und mit Wert aus Register (R1 =1) addieren wieder auf Register 1 schreiben

0	1	H	-
		R	$Z := \mathbf{2}, CAR++$
		B	$z \rightarrow MAR$
	Lesen, 1 Byte		
	2	H	$[MAR] \rightarrow MDR \rightarrow Y, R_1 \rightarrow X$
		R	$Z := X + Y, CAR++$
		B	$Z \rightarrow R_0$

Beispielflowchart

Min{R0, R1} auf RAM [9]