Rekursives Schaltungsdesign

Rekursives Schaltungsdesign

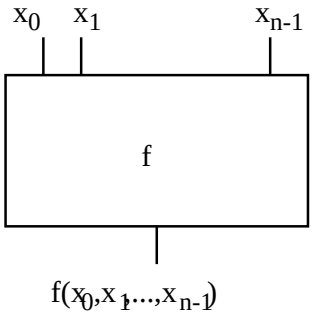
Motivation: aus einfachen Dingen, komplizierte herstellen.

Digitale Zustände: es gibt nur Null und Eins und die *Zeit*. Null und Eins muss interpretiert und realisiert werden. Entweder *high* oder *low* wird als logische Null oder Eins interpretiert (*high*=1: positive Logik, ‘active high logic’); *high*=0: negative Logik, ‘active low logic’).

Schaltfunktionen

Frischen wir erst Altbekanntes auf.

Wir interessieren uns für Schaltnetze mit nur einem Ausgang und *n* Eingängen, die wir mit Indizes durchnummerieren. Jeder Eingang und auch der Ausgang soll nur einen der beiden Zustände annehmen können. Der Ausgang soll *allein durch die Eingangsbelegung* festgelegt sein.



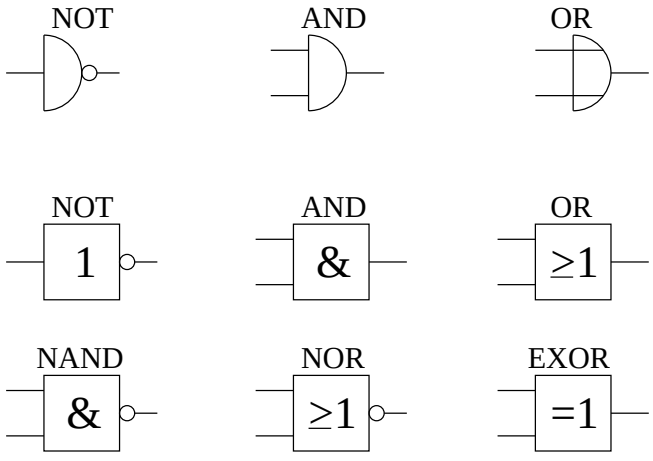
Einfache Schaltfunktionen mit zwei Eingängen sind die UND-, die ODER- und die NICHT-Funktion:

Konjunktion	UND-Funktion	$\cdot : \{0,1\}^2 \longrightarrow \{0,1\}$
Disjunktion	ODER-Funktion	$+ : \{0,1\}^2 \longrightarrow \{0,1\}$
Negation	NICHT-Funktion	$\_ : \{0,1\} \longrightarrow \{0,1\}$

Wertetabellen der Funktionen:

x_0	x_1	$x_0 \cdot x_1$	$x_0 + x_1$	$\overline{x_0}$
0	0	0	0	1
0	1	0	1	
1	0	0	1	0
1	1	1	1	

Man realisiert diese Schaltfunktionen durch sogenannte logische Gatter oder gates mit den folgenden Schaltsymbolen:

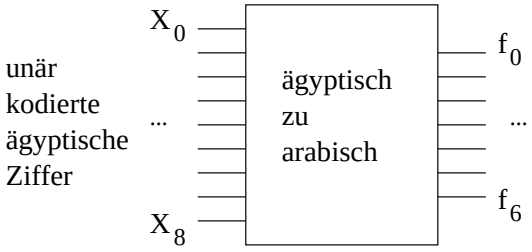


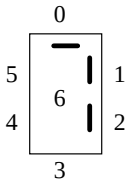
Alle Schaltfunktionen, die mit zwei Eingängen zu realisieren sind zeigt folgende Tabelle (einige sind auch als Symbole oben dargestellt, hier erscheinen die englischen Namen):

X_0	X_1	ZERO	AND		X_0		X_1	XOR	OR
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

X_0	X_1	NOR	EQUI	$\overline{X_1}$		$\overline{X_0}$		NAND	ONE
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Betrachten wir zur Auffrischung der Kenntnisse die Konstruktion einer einfachen Schaltung. Wir wollen mithilfe der ägyptischen Zahlendarstellung (hier einer geordneten Unär-Darstellung) eine moderne Siebensegmentanzeige ansteuern (dabei verzichten wir auf die Null).





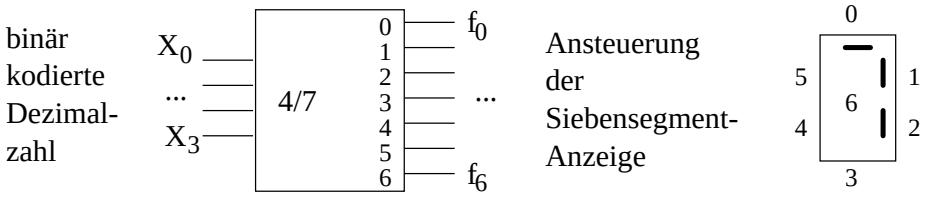
Man erhält mit den Vereinbarungen aus der Abbildung folgende Wertetabelle:

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	f_0	f_1	f_2	f_3	f_4	f_5	f_6
1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0	0	1	1	0	1	1	0	1
1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	1
1	1	1	1	0	0	0	0	0	0	1	1	0	0	1	1
1	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1
1	1	1	1	1	1	0	0	0	1	0	1	1	1	1	1
1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
alle anderen 503 Mögl.									alles 0						

Damit ergibt sich z.B. die VDNF (vollständige disjunktive Normalform) für die Funktion zur Ansteuerung des oberen Segmentelements zu:

$$
\begin{aligned}
f_0(X) \equiv\ & X_0X_1\,\overline{X_2}\,\overline{X_3}\,\overline{X_4}\,\overline{X_5}\,\overline{X_6}\,\overline{X_7}\,\overline{X_8} + X_0X_1X_2\overline{X_3}\,\overline{X_4}\,\overline{X_5}\,\overline{X_6}\,\overline{X_7}\,\overline{X_8} + \\
& X_0X_1X_2X_3X_4\overline{X_5}\,\overline{X_6}\,\overline{X_7}\,\overline{X_8} + X_0X_1X_2X_3X_4X_5\overline{X_6}\,\overline{X_7}\,\overline{X_8} + \\
& X_0X_1X_2X_3X_4X_5X_6\overline{X_7}\,\overline{X_8} + X_0X_1X_2X_3X_4X_5X_6X_7\overline{X_8} + \\
& X_0X_1X_2X_3X_4X_5X_6X_7X_8
\end{aligned}
$$

Die weiteren Funktionen erhält man auf analoge Art und Weise. Eine VKNF (vollständige konjunktive Normalform) wäre an dieser Stelle wegen der vielen Nullen sehr teuer. Die heute gebräuchlichere Umsetzung geht von einer Binär-Darstellung (sogenannter binär kodierter Dezimalzahlen oder einfach BCD-Zahlen) aus. Man benötigt 4 Bit, kann damit aber sogar 16 Ziffern kodieren. Wir nehmen an, dass nur 10 Ziffern umgesetzt werden (eine Erweiterung auf die Hexadezimalzahlen ist einfach), die nicht erlaubten Ziffern dürfen eine beliebige Anzeige erzeugen.



Die Wertetabelle (nun einschließlich einer Null!) sieht wie folgt aus:

X_0	X_1	X_2	X_3	f_0	f_1	f_2	f_3	f_4	f_5	f_6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	hier kann man frei wählen						
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

Da oft weniger 0en als 1en vorkommen, empfehlen sich für viele der Funktionen die VKNFs, z.B. (man wählt die freien Einträge entsprechend als 1en):

$$f_2(X) \equiv X_0 + X_1 + \overline{X_2} + X_3$$

Rekursives Schaltungsdesign

Wir gehen im Folgenden nicht auf Minimierungsverfahren zur Reduktion der Kosten eines Schaltkreises ein. Zu den bekannten Verfahren gehören die Methoden von
Quine-McClusky oder die Veitch- bzw.
Karnaugh-Diagramme. Vielmehr konzentrieren wir uns auf die rekursive Konstruktion regelmäßiger Funktionseinheiten.

Wiederholen wir aber zuerst, was unter Kosten und Tiefe der Realisierung einer Schaltfunktion zu verstehen ist.

Kosten und Tiefe

Wir haben Schaltfunktionen auf zwei Arten realisiert:

- durch Formeln (eigentlich boolesche Ausdrücke)
- durch Schaltkreise

Für die Kosten *C* vereinbaren wir z.B. folgendes:

Formel
    Summe der vorhandenen Operationszeichen, also +,.,_
Schaltkreis
    Summe der vorhandenen Gatter

Die Kosten sollen in irgend einer Art und Weise reale Kosten widerspiegeln, z.B. Anzahl der Chips, Preis der Chips, Platzverbrauch der Chips, Verlustleistung der Chips usw. Durch entsprechende Anpassung der Kostenfunktion künnen die realen Kosten abgeschätzt werden.

Für die Tiefe *T* vereinbaren wir z.B. folgendes:

Formel

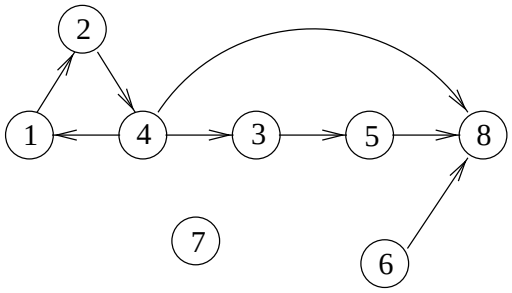
Tiefe der Klammerverschachtelung, wenn wir die Formel vollständig klammern würden. Ist der Ausdruck vollständig geklammert, kann die Tiefe durch einfaches Bestimmen der Tiefe der öffnenden Klammern durch einen Lauf über den Ausdruck berechnet werden.

Schaltkreis

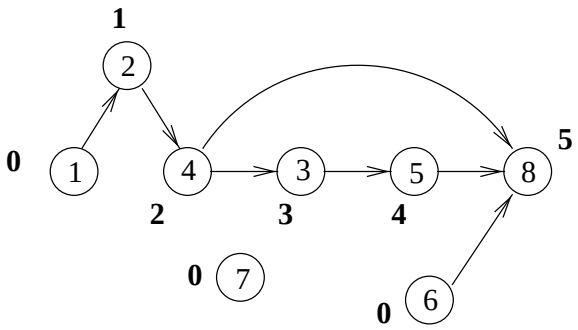
Tiefe des Graphen, wenn wir die Verschaltung der Gatter als zykelfreien Graphen interpretieren. Sind die Knoten des Graphen topologisch sortiert, kann die Tiefe durch einen in der Anzahl der Knoten und Kanten (Pfeile) linearen Algorithmus bestimmt werden. (Das topologische Sortieren der Knoten geht auch in Linearzeit.)

Hierzu zwei Beispiele:

Nicht zykelfreier Graph (damit keine Repräsentation eines Schaltkreises):



Zykelfreier Graph mit Angabe der Tiefe der Knoten:



Eine topologische Sortierung der Knoten ist z.B. gegeben durch: 1,7,6,2,4,3,5,8.

Beachte: bei diesen Definitionen für Kosten und Tiefe wurden die Verbindungen zwischen den Gattern nicht berücksichtigt!

Decoder

Ein Decoder ist ein Schaltkreis, der je nach Wert (Interpretation als Binärzahl) am Eingang, genau einen Ausgang auf 1 setzt.

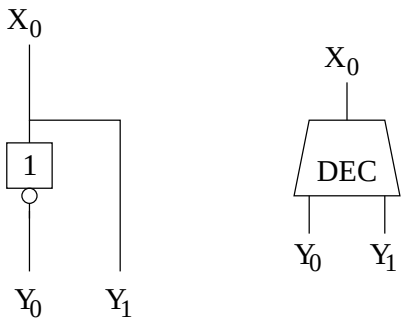
Schaltfunktion $DEC : \{0,1\} \longrightarrow \{0,1\}^2$ mit

$$DEC(X_0) = (Y_0, Y_1) \quad \text{mit} \quad Y_i = \begin{cases} 1 & \text{falls} \quad \langle X_0 \rangle = i \\ 0 & \text{sonst} \end{cases}, i = 0,1$$

Wertetabelle:

X_0	Y_0	Y_1
0	1	0
1	0	1

Man sieht sofort: $Y_0 = \overline{X_0}, Y_1 = X_0$



Schaltfunktion $DEC_n : \{0,1\}^{\log n} \longrightarrow \{0,1\}^n$ mit

$$DEC_n(X_0, \ldots, X_{\log n-1}) = (Y_0, \ldots, Y_{n-1})$$

wobei für alle $i = 0, \ldots, n-1$ gilt:

$$Y_i = \begin{cases} 1 & \text{falls} \quad \langle X_{\log n-1} \ldots X_0 \rangle = i \\ 0 & \text{sonst} \end{cases}$$

Damit ist der DEC von oben ein DEC_2.
Wertetabelle für $n = 8$:

X_2	X_1	X_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

VDNFs für DEC_8:

$$\begin{aligned} Y_0 &= \overline{X_0}\,\overline{X_1}\,\overline{X_2} \\ Y_1 &= \overline{X_0}\,\overline{X_1}X_2 \\ &\ldots \\ Y_7 &= X_0X_1X_2 \end{aligned}$$

Realisierung mit Normalformen

Man kann nun den Decoder mithilfe der Normalformen realisieren, wobei sich für *n=8* die Kosten zu *2*8+8*3/2=28* ergeben. Allgemein gilt für die Kosten:

$$C(DEC_n) = n \cdot (\log n - 1) + n/2 \cdot \log n = n \log n - n/2 + \log n$$

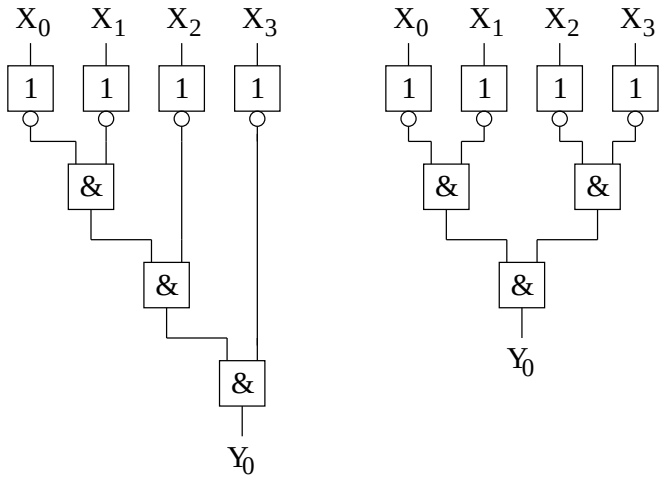
da jeder der *n* Minterme *log(n)-1* viele UND-Gatter und insgesamt genauso viele NICHT-Gatter wie Nullen auf der linken Seite der Tabelle vorkommen.

Die Anzahl der Inverter kann reduziert werden, da jede Schaltvariable nur einmal negiert werden muss; es sind also lediglich *log(n)* viele NICHT-Gatter erforderlich. Also:

$$C(DEC_n) = n \cdot (\log n - 1) + \log n = n \log n - n + \log n$$

Wie tief ist diese Konstruktion?

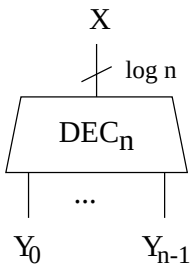
Es kommt darauf an, wie geschickt wir die Minterme realisieren. Bei vier Variablen gibt es z.B. die folgenden Möglichkeiten:



Wir können also eine Tiefe von 3 erhalten. Allgemein erhalten wir für die Tiefe eines *n*-Decoders:

$$T(DEC_n) = \log\log n + 1$$

Schaltsymbol des *n*-Decoder (d.h. die Größe wird durch die Anzahl der Ausgänge bestimmt):

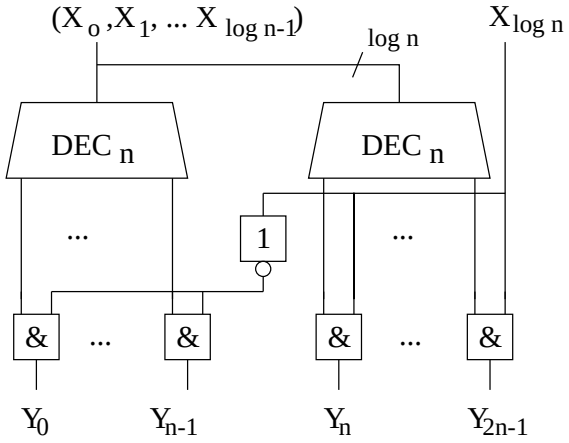


Wir haben also viele verschiedene Darstellungen für den gleichen Sachverhalt, nämlich der Beschreibung einer Schaltfunktion:

- Beschreibung in Worten
- Mathematische Beschreibung als Funktion
- Wertetabelle
- Formel (boolescher Ausdruck)
- Schaltkreis
- Schaltsymbol

Rekursive Realisierung

Nehmen wir an, wir hätten schon einen *n*-Decoder realisiert. Wir wollen nun mit zwei solchen und einigen weiteren Gattern einen *2n*-Decoder zusammenbauen.



(Bemerkung: diese Schaltung ist offensichtlich nicht die schlauste; man kann auch einfach nur einen *n*-Decoder verwenden, aber trotzdem ist sie korrekt und wir können Kosten und Tiefe berechnen. Später sehen wir die einfachere Schaltung.) Wie berechnen wir nun aber Kosten und Tiefe?

Rekursionsformeln

Wir lernen hier zwei einfache Rekursionsgleichungen kennen, die uns sehr nützlich sein werden. Die Idee dahinter ist recht einfach zu verstehen (hier sehr informal):

Gegeben ein Problem der Großenordnung *n* (was immer das bedeuten mag), kennt man die Lösung des Problems kleinerer Ordnung sowie eine Konstruktion, wie man mithilfe dieser kleineren Lösung, die Lösung für das große Problem erhält, so kann man den Aufwand mithilfe der Rekursionsgleichung berechnen. Das kleinere Problem kann einfach von der Größenordnung *n-1* sein (lineare Rekursion), oder aber lediglich von der Größenordnung *n/b*, d.h. dem *b*-ten Teil, sein (logarithmische Rekursion).

Verwenden wir die folgenden Bezeichnungen:

f(n)
: ist die gesuchte Funktion;

a
: gibt an, wie oft das kleinere Problem benutzt wird;

b
: gibt an, um wieviel kleiner das kleinere Problem ist (bei der logarithmischen Rekursion);

c
: gibt die Größe des kleinsten Problems (i.A. *f(1)*) an;

g(n)
: gibt an, was bei jedem Schritt zusätzlich an Aufwand in Abhängigkeit von *n* getrieben werden muss.

Formeln für die lineare Rekursion:

Sei $f : \mathbb{N} \longrightarrow \mathbb{N}$ eine Funktion mit:

$$\begin{aligned} f(1) &= c \qquad \text{für irgendein } c \in \mathbb{N} \\ f(n) &= a \cdot f(n-1) + g(n) \end{aligned}$$

Dann gilt:

$$f(n) = a^{n-1} \cdot c + \sum_{i=0}^{n-2} a^i \cdot g(n-i)$$

Formeln für die logarithmische Rekursion:

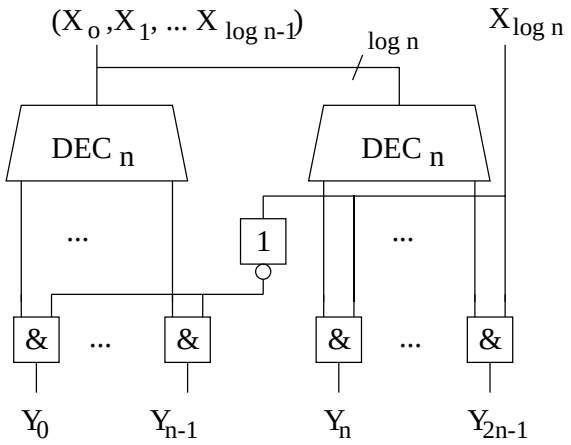
Sei $f : \mathbb{N} \longrightarrow \mathbb{N}$ eine Funktion mit:

$$\begin{aligned} f(1) &= c \qquad \text{für irgendein } c \in \mathbb{N} \\ f(n) &= a \cdot f\left(\frac{n}{b}\right) + g(n) \end{aligned}$$

für alle Potenzen $n = b^k$ (d.h. uns interessiert die Funktion f nur an den Stellen $b_0 = 1, b, b^2, b^3, \ldots$).
Dann gilt:

$$f(n) = a^{\log_b n} \cdot c + \sum_{i=0}^{\log_b n - 1} a^i \cdot g\left(\frac{n}{b^i}\right)$$

Betrachten wir nochmals den oben bereits gezeigten *2n*-Decoder:



Die Parameter für die logarithmische Rekursionsgleichung ergeben sich zu:

a	2	da 2 DEC_n
b	2	da n die Hälfte von $2n$
c	1	da $f(1) = C(DEC_2) = 1$ (besteht nur aus einem Inverter)
$g(n)$	$2n+1$	$2n$ UND-Gatter und 1 Inverter

Für die Kosten des rekursiv definierten *2n*-Decoders erhalten wir also:

Die Funktion f ergibt sich für den Decoder als:

$$\begin{aligned} f(1) &= C(DEC_2) = 1 \\ f(n) &= C(DEC_{2n}) \\ &= 2 \cdot C(DEC_n) + 2n + 1 \end{aligned}$$

und wir erkennen die Konstanten sowie die Funktion g als

$$a = 2 \qquad b = 2 \qquad c = 1 \qquad g(n) = 2n + 1$$

womit sich die Kosten eines $2n$-Decoders wie folgt berechnen

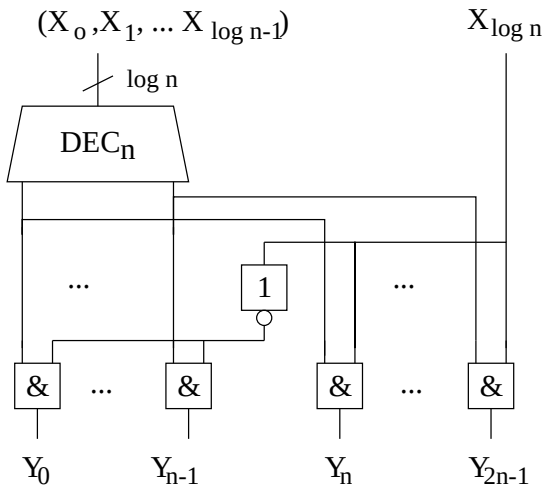
$$\begin{aligned} f(n) &= a^{\log_b n} \cdot c + \sum_{i=0}^{\log_b n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) \\ &= 2^{\log_2 n} \cdot 1 + \sum_{i=0}^{\log_2 n - 1} 2^i \cdot \left(\frac{2n}{2^i} + 1\right) \\ &= n + \sum_{i=0}^{\log n - 1} 2n + \sum_{i=0}^{\log n - 1} 2^i \\ &= n + 2n \log n + 2^{\log n} - 1 \\ &= 2n \log n + 2n - 1 \\ &= 2n \log(2n) - 1 \end{aligned}$$

Und somit als Kosten eines n-Decoders:

$$C(DEC_n) = n \log n - 1$$

Es ist eine schöne Übung, das Ergebnis konkret für einen *4*- und *8*-Decoder nachzurechnen.

Nun haben wir schon bemerkt, dass die Konstruktion nicht die geschickteste war. Wir können ja einfach einen der beiden *n*-Decoder weglassen. Tun wir dies.



Die Parameter für die logarithmische Rekursionsgleichung sind damit wie folgt:

a	1	da 1 DEC_n
b	2	da n die Hälfte von $2n$
c	1	da $f(1) = C(DEC_2) = 1$
		(besteht nur aus einem Inverter)
$g(n)$	$2n+1$	$2n$ UND-Gatter und 1 Inverter

und die Kosten berechnen sich nun zu:

$$\begin{aligned}
f(n) &= C(DEC_{2n}) \\
&= a^{\log_b n} \cdot c + \sum_{i=0}^{\log_b n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) \\
&= 1^{\log_2 n} \cdot 1 + \sum_{i=0}^{\log_2 n - 1} 1^i \cdot \left(\frac{2n}{2^i} + 1\right) \\
&= 1 + \sum_{i=0}^{\log n - 1} \frac{2n}{2^i} + \sum_{i=0}^{\log n - 1} 1 \\
&= 1 + 2n \cdot \sum_{i=0}^{\log n - 1} \left(\frac{1}{2}\right)^i + \log n \\
&= 1 + 2n \cdot \frac{(1/2)^{\log n} - 1}{1/2 - 1} + \log n \\
&= 1 + 2n \cdot (1/2^{\log n} - 1) \cdot -2 + \log n \\
&= 1 - 4n \cdot (1/n - 1) + \log n \\
&= 4n + \log n - 3
\end{aligned}$$

womit sich für einen *n*-Decoder Kosten ergeben, die um einen Faktor *log(n)* niedriger liegen als in der vorhergehenden Konstruktion:

$$C(DEC_n) = 2n + \log n - 4$$

$$(\text{wegen } C(DEC_{2n}) = 4n + \log n - 3 = 2 \cdot 2n + \log(2n) - 4)$$

Schauen wir uns nun die Tiefe unserer Konstruktionen an. Die Tiefen beider Konstruktionen sind offensichtlich identisch. Berechnen wir die Tiefe der Decoder ebenfalls mit der Rekursionsformel.

Nach Konstruktion gilt offensichtlich

$$T(DEC_{2n}) = \max(T(DEC_n), 1) + 1$$

nun hat ein Decoder mindestens Tiefe 1, also können wir das Maximum weglassen:

$$T(DEC_{2n}) = T(DEC_n) + 1$$

womit sich die Tiefe eines *2n*-Decoders mit folgenden Parametern

a	1	da DEC_n einmal durchlaufen wird
b	2	da n die Hälfte von $2n$
c	1	da $f(1) = T(DEC_2) = 1$
		(besteht nur aus einem Inverter)
$g(n)$	1	da nur 1 UND-Gatter mehr durchlaufen wird

wie folgt berechnet

$$
\begin{aligned}
f(n) &= T(DEC_{2n}) \\
&= a^{\log_b n} \cdot c + \sum_{i=0}^{\log_b n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) \\
&= 1^{\log_2 n} \cdot 1 + \sum_{i=0}^{\log_2 n - 1} 1^i \cdot 1 \\
&= 1 + \sum_{i=0}^{\log n - 1} 1 \\
&= \log n + 1
\end{aligned}
$$

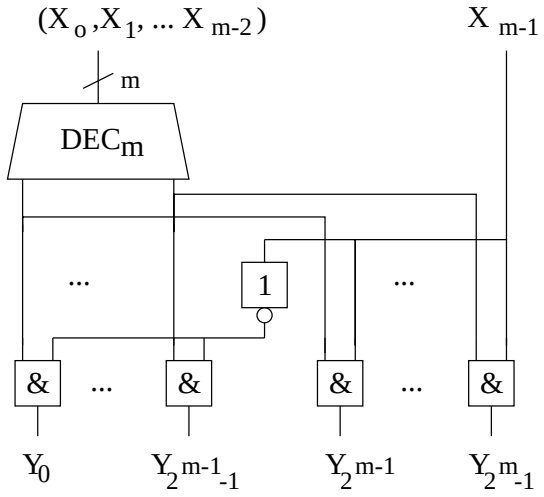
Die Tiefe eines *n*-Decoders beträgt also

$$T(DEC_n) = \log n$$

was deutlich mehr als die *log(log(n))* von der Konstruktion mit der Normalform ist.

Geben wir uns also damit zufrieden? Nein!

In obigen Konstruktionen haben wir die Größe der Decoder bezüglich der Ausgänge bestimmt. Wiederholen wir kurz die Diskussion, nun allerdings bestimmen wir die Größe bezüglich der Eingänge.



Nun nutzen wir die Formel der linearen Rekursion und erhalten mit den Parametern:

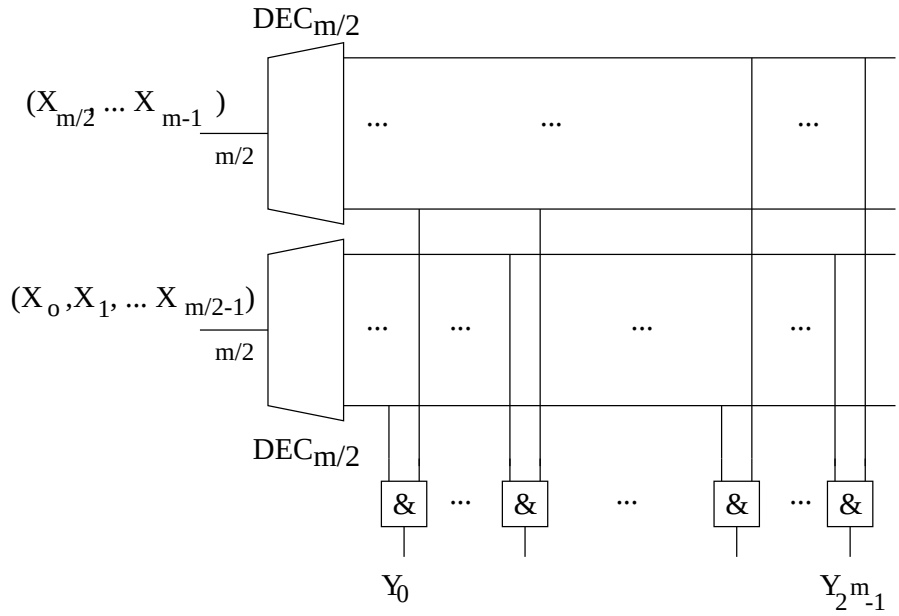
a	1	da 1 DEC_{m-1}
c	1	da $f(1) = C(DEC_1) = 1$ (besteht nur aus einem Inverter)
$g(m)$	$2^m + 1$	2^m UND-Gatter und 1 Inverter

folgende Kosten:

$$\begin{aligned} f(m) &= a^{m-1} \cdot c + \sum_{i=0}^{m-2} a^i \cdot g(m-i) \\ &= 1^{m-1} \cdot 1 + \sum_{i=0}^{m-2} 1^i \cdot 2^{m-i} + 1 \\ &= 1 + \sum_{i=0}^{m-2} 2^{m-i} + 1 \\ &= 2 \cdot 2^m + m - 4 \end{aligned}$$

Wie nicht anders zu erwarten, erhalten wir die gleichen Kosten wie oben, wenn wir für *m* wieder *log(n)* einsetzen.

Nun stellt sich die Frage, ob nicht auch eine logarithmische Konstruktion möglich ist. Nehmen wir an *m* sei eine Zweierpotenz.



Die Parameter der Rekursionsgleichung für die Kosten ergeben sich als:

a	2	da 2 $DEC_{m/2}$
b	2	da Decoder halber Größe
c	1	da $f(1) = C(DEC_1) = 1$ (besteht nur aus einem Inverter)
$g(m)$	2^m	da 2^m UND-Gatter

und damit erhalten wir die folgende Gleichung

$$
\begin{aligned}
f(m) &= C(DEC_m) \\
&= a^{\log_b m} \cdot c + \sum_{i=0}^{\log_b m - 1} a^i \cdot g\left(\frac{m}{b^i}\right) \\
&= 2^{\log_2 m} \cdot 1 + \sum_{i=0}^{\log_2 m - 1} 2^i \cdot 2^{m/2^i} \\
&= m + \sum_{i=0}^{\log m - 1} 2^{m/2^i - i}
\end{aligned}
$$

die allerdings nicht ganz so einfach zu einer einfachen geschlossenen Formel aufzulösen ist. Allerdings kann man zeigen, dass für wachsende *m* gilt:

$$\frac{f(m)}{2^m} \longrightarrow 1$$

Die verbesserte Konstruktion kostet also für hinreichend große *m* ungefähr die Häfte.

Betrachten wir nun die Tiefe der neuen Konstruktion. Die Parameter der Rekursionsgleichung für die Tiefe ergeben sich als:

a	1	da ein $DEC_{m/2}$ durchlaufen wird
b	2	da Decoder halber Größe
c	1	da $f(1) = T(DEC_1) = 1$
		(besteht nur aus einem Inverter)
$g(m)$	1	da 1 UND-Gatter mehr

und wir erhalten die gleiche Tiefe wie bei der Konstruktion mit der Normalform (allerdings bei erheblich geringeren Kosten):

$$\begin{aligned} f(m) &= T(DEC_m) \\ &= a^{\log_b m} \cdot c + \sum_{i=0}^{\log_b m - 1} a^i \cdot g\left(\frac{m}{b^i}\right) \\ &= 1^{\log_2 m} \cdot 1 + \sum_{i=0}^{\log_2 m - 1} 1^i \cdot 1 \\ &= 1 + \sum_{i=0}^{\log m - 1} 1 \\ &= \log m + 1 \end{aligned}$$

Fassen wir die Ergebnisse der verschiedenen Konstruktionen nochmals zusammen (wir betrachten wieder als Grundlage für die Größe die Anzahl der Ausgänge):

	Tiefe	Kosten	H-Drähte
Normalform	$\log\log n + 1$	$n\log n - n + \log n$	$2\log n + \log\log n - 1$
1. Konst.	$\log n$	$n\log n - 1$	$\log n(\log n + 1)/2$
2. Konst.	$\log n$	$2n + \log n - 4$	$n + \log n - 2$
3. Konst.	$\log\log n + 1$	$n + x$	$2\sqrt{n} + x$

Vergleicher

Wenden wir die rekursive Konstruktionsmethode an, um Vergleicher zu bauen.

Ein Vergleicher ist ein Schaltkreis, der zwei Eingangssignale miteinander vergleicht und genau einen Ausgang auf 1 setzt, je nachdem, ob der Vergleich kleiner, gleich oder größer ergibt.

$$\text{Schaltfunktion } CMP : \{0,1\}^2 \longrightarrow \{0,1\}^3 \text{ mit}$$

$$CMP(X_0, Y_0) = (Z_0, Z_1, Z_2)$$

wobei gilt:

$$Z_0 = \begin{cases} 1 & \text{falls} \quad \langle X_0 \rangle < \langle Y_0 \rangle \\ 0 & \text{sonst} \end{cases}$$

$$Z_1 = \begin{cases} 1 & \text{falls} \quad \langle X_0 \rangle = \langle Y_0 \rangle \\ 0 & \text{sonst} \end{cases}$$

$$Z_2 = \begin{cases} 1 & \text{falls} \quad \langle X_0 \rangle > \langle Y_0 \rangle \\ 0 & \text{sonst} \end{cases}$$

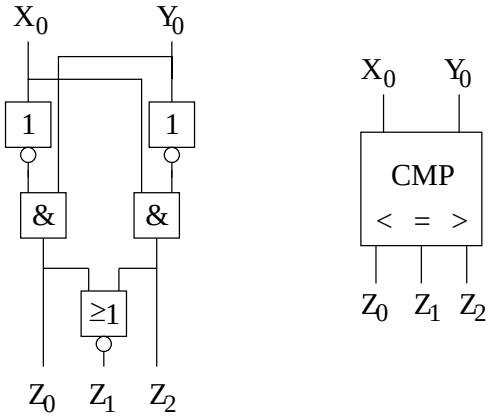
Dies ergibt folgende Wertetabelle:

X_0	Y_0	Z_0	Z_1	Z_2
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Wir sehen sofort die Schaltfunktionen sowie eine mögliche Realisierung mit den sich ergebenden Kosten und Tiefe:

$$Z_0 = \overline{X_0} Y_0 \qquad Z_2 = X_0 \overline{Y_0} \qquad Z_1 = \overline{Z_0 + Z_2}$$

$$\begin{aligned} C(CMP) &= 5 \\ T(CMP) &= 3 \end{aligned}$$



Will man zwei *n*-stellige Binärdarstellungen zweier Zahlen miteinander vergleichen, muss man folgende Schaltfunktion realisieren.

Schaltfunktion $CMP_n : \{0,1\}^{2n} \longrightarrow \{0,1\}^3$ mit

$$CMP(X_0, \ldots, X_{n-1}, Y_0, \ldots, Y_{n-1}) = (Z_0, Z_1, Z_2)$$

wobei gilt:

$$Z_0 = \begin{cases} 1 & \text{falls} \quad \langle X_{n-1} \ldots X_0 \rangle < \langle Y_{n-1} \ldots Y_0 \rangle \\ 0 & \text{sonst} \end{cases}$$

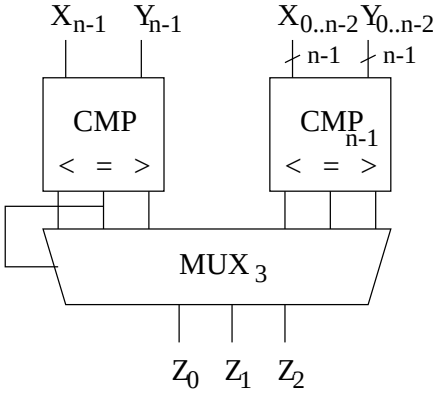
$$Z_1 = \begin{cases} 1 & \text{falls} \quad \langle X_{n-1} \ldots X_0 \rangle = \langle Y_{n-1} \ldots Y_0 \rangle \\ 0 & \text{sonst} \end{cases}$$

$$Z_2 = \begin{cases} 1 & \text{falls} \quad \langle X_{n-1} \ldots X_0 \rangle > \langle Y_{n-1} \ldots Y_0 \rangle \\ 0 & \text{sonst} \end{cases}$$

Nun gilt aber offensichtlich folgendes (schreiben wir hierzu abkürzend für die beiden *n*-stelligen Eingänge *X* und *Y*):

X ist kleiner als *Y*, wenn bereits die oberste Stelle kleiner ist. *X* ist größer als *Y*, wenn bereits die oberste Stelle größer ist. Sind die beiden obersten Stellen gleich, so entscheiden allein die unteren Stellen.

Wir können also einen *n*-Vergleicher wie folgt aufbauen:



Wir wenden die Formel der linearen Rekursion an und erhalten mit den Parametern für die Kosten und die Tiefe:

	C	T
a	1	1
c	5	3
$g(n)$	15	3

die folgenden Kosten bzw. Tiefe:

$$\begin{aligned} C(CMP_n) &= 1^{n-1} \cdot 5 + \sum_{i=0}^{n-2} 1^i \cdot 15 \\ &= 5 + (n-1) \cdot 15 \\ &= 15n - 10 \end{aligned}$$

$$\begin{aligned} T(CMP_n) &= 1^{n-1} \cdot 3 + \sum_{i=0}^{n-2} 1^i \cdot 3 \\ &= 3 + (n-1) \cdot 3 \\ &= 3n \end{aligned}$$

Beachte: auch hier konnte man das eigentlich notwendige Maximum über beide Pfade eliminieren, da stets gilt:

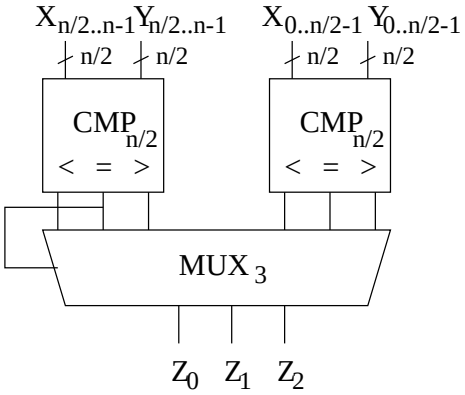
$$T(CMP_{n-1}) \geq T(CMP)$$

Die obige Konstruktion benutzt nur das oberste Bit für die Rekursion. Es gilt aber doch auch die allgemeinere Form:

$$\langle X \rangle < \langle Y \rangle \text{ falls } \langle X_{n-1} \dots X_{n-j} \rangle < \langle Y_{n-1} \dots Y_{n-j} \rangle$$

für ein beliebiges $j = 1, \dots, n$.

Die anderen Fälle für gleich und größer gelten analog. Lassen wir für *n* nur Zweierpotenzen zu, so können wir den *n*-Vergleicher durch folgende Konstruktion realisieren:



Das heißt, wir vergleichen zuerst die oberen und die unteren Hälften der Eingabewerte und entscheiden anschließend über das Resultat. Für die Kosten und die Tiefe ergeben sich nun mit den Parametern

	C	T
a	2	1
b	2	2
c	5	3
$g(n)$	10	3

die folgenden Lösungen (jetzt logarithmische Rekursion):

$$\begin{aligned} C(CMP_n) &= 2^{\log n} \cdot 5 + \sum_{i=0}^{\log n - 1} 2^i \cdot 10 \\ &= 5 \cdot n + 10 \cdot (n-1) \\ &= 15n - 10 \end{aligned}$$

$$\begin{aligned} T(CMP_n) &= 1^{\log n} \cdot 3 + \sum_{i=0}^{\log n - 1} 1^i \cdot 3 \\ &= 3 + 3 \cdot \log n \end{aligned}$$

Die Kosten haben sich also nicht verändert, was wir auch genau so erwartet haben. Die Tiefe hat sich jedoch von linearer auf logarithmische Größenordnung reduziert.

Mit der gleichen Methode lassen sich auch viele andere regelmäßige Basisfunktionseinheiten aufbauen. Beispiele sind:

- Demultiplexer
- Encoder
- Führende Null-Zähler
  (gibt binär kodiert die Anzahl der führenden Nullen in einer Bitfolge an)
- Populationcounter
  (gibt binär kodiert die Anzahl der Einsen in einer Bitfolge an)

CVS: Id

formella@cs.uni-sb.de

http://www-wjp.cs.uni-sb.de/~formella/izfp.html
