MULTIPLIZIERER mit binären Daten

Robert SCHWAGER

Einleitung

Multiplizier- und Dividiereinheiten werden heutzutage als eine hardwaremäßige Einheit in moderne digitale Computer - insbesondere in schnelle Arithmetikprozessoren- mit IC-Technologie eingesetzt. Das Operationszeichen eines Multiplizierers wird durch ein „**x**" gekennzeichnet.

$$\mathbf{P = A \times B}$$

Der Multiplikand **A** bestehend aus *m*-Bits, der Multiplikator **B** bestehend aus *n*-Bits und ergeben ein Multiplikationsprodukt **P** aus *(m+n)*-Bits. Besitzen die beiden Faktoren **A** und **B** je ein zusätzliches Vorzeichenbit, so ist das Ergebnis des Produktes **P** nur *(m+n+1)*-Bit breit.

Grundlagen zu Rechnen mit Dualzahlen - Multiplikation

Die beiden Produkte **A** und **B** können als Summe ausgedrückt werden:

$$A_r = a_{m-1}..a_1a_0 = \sum_{i=0}^{m-1} a_i 2^i \qquad B_r = b_{n-1}..b_1b_0 = \sum_{j=0}^{m-1} b_j 2^j$$

Das Produkt **P** kann man daher auch als „Doppel"-Summe ausdrücken:

$$P_v = A_v \times B_v = (\sum_{i=0}^{m-1} a_i 2^i) \times (\sum_{j=0}^{n-1} b_j 2^j) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} (a_i \& b_j) = \sum_{k=0}^{m+n-1} P_k 2^k$$

Die Konstante $\mathbf{2^f}$ (für *f i* bzw. *j*) stellt die Stellenposition *f* dar

(*f=0*..Bit 0 ... *f=x*..Bit x).

Beschreibung der Nomenklatur: $\chi = \alpha\,\beta$ .. α stellt das obere Hälfte des Argument von χ und β die untere Hälfte des Argument von χ dar. Für obere und untere Hälfte kann auch der Index L und H geschrieben werden.

($\cap$ bzw. $\wedge$ bzw.) **&**.. logisch UND

($\cup$ bzw. $\vee$ bzw.) **|**.. logisch ODER

+ .. Addition
- .. Subtraktion
x .. Multiplikation
exp (x) = e^x .. Exponentialfunktion von x
ln (x) ... natürlicher Logarithmus von x

Multiplizierer-Hardware und Algorithmus

Standardmultiplizierer

1. indirekte Multiplikation (Algorithmus und Hardware)

Ein typischer indirekter Multiplizierereinheit besteht zunächst aus 3 n-Bit parellelgeladene Register - Akkumulator **AC**, Multiplikator-Register **MR** und Hilfsregister **AX** (Auxillary Reg.). Der Multiplikand **A** wird in das **AX**-Register, der Multiplikator **B** in das **MR**-Register und *Nullen* in das **AC**-Register geladen. Das nebenstehende Bild, soll den grundsätzlichen Aufbau zeigen. Die größte Aufgabe hat der n-Bit-Addierer, der eigentlich den Hauptteil dieser Einheit ist. Das log. UND-Glied verknüpft das **AX**-Register (**A**) und den jeweiligen Bit des **MR** (**B**), das Ergebnis ist das **AX** bzw. **0**. Der Addierer kann als kaskadierbarer Schiebeaddierer realisiert werden.

$$\mathbf{C_{out}\ S = (\ AC\ ) + (\ AX \wedge MR_n\ ) + C_{in}}$$

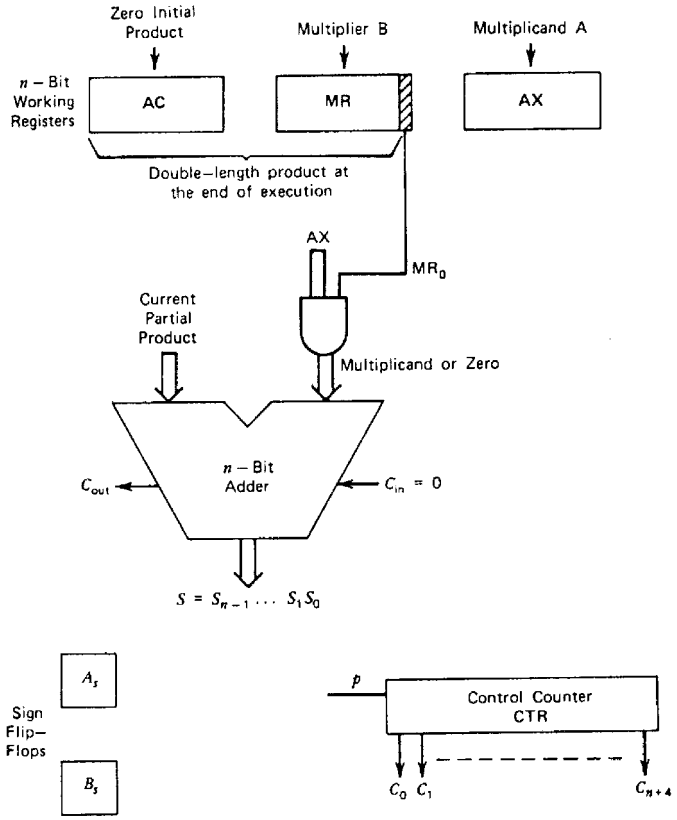

Abbildung 1: Schematischer Ausbau eines indirekten Multiplizierers

2. indirekter universal Multiplizierer

Zwei externe Eingänge (**X Y**) kontrollieren die Datensignale *US* (**0 0**) - unsign - kein Vorzeichen- , *SM* (**0 1**) - signmagnitude - Zeichengröße, *OC* (**1 0**) - one's complement - Einerkomplement - und *TC* (**1 1**)- Two's Complement - Zweierkomplement. Die Hardwarebeschreibung ist der des in **1**. erklärtem Multiplizierer ident, jedoch nur die Register werden modifiziert.

3. vielfacher Schiebe-Multiplizierer

Die Multiplikationsdurchführung in Prozessen kann beschleunigt werden, wenn mehr als ein Muliplizierbit pro Zyklus verwendet wird. Und z.B. die gesamte Anzahl der Addierer um die Hälfte zu reduziert, die Initialisierungsregister werden gekennzeichnet (wie zuvor). Dieser nichtüberlappende "absuchende" Multiplikation wird durch einen CARRY-SAVE-ADDIERER (*CSA*) (übertragsspeichernder Addierer) und einem CARRY-PROPGATE-ADDER (*CPA*) (übertragfortpflanzenden Addierer) ergänzt. Im Bild (Abb.2) wird ein 2-stelliger Multiplikand dargestellt. Der *CSA* hat nun die Aufgabe die log. UND-Verknüpfungen des **A** und des einzelnen **B** zu addieren, sowie des **AC**, wobei die MR_n die Stelle (2n) angibt. Der *CSA* hat somit *n+1*-Eingänge. Der *CPA* hat die Aufgabe die Summe von **AC** und das Ergebnis vom *CSA* zu bilden.

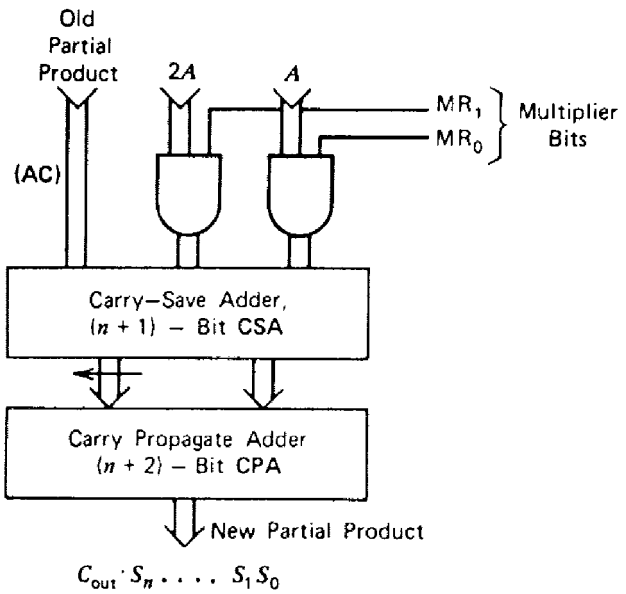

Abb.2: **Schematische Darstellung eines vielfacher Schiebe-Multiplizierers**

4. Multiplikation durch überlappende, mehrmalige Bit-Überprüfung

Die Idee basiert auf Nullen in der Multiplikation. Je größer die Anzahl der Nullen im Multiplikator ist, so schneller ist dieser Multiplizierer. Die Funktionsweise ist recht simpel. Die Bits des Multiplikators **B** sind die Steuerbit. Es wird beim niedrigsten Bit von **B** begonnen. Ist das x-te Bit des Multiplikators **B** *NULL* so wird das Zwischenergebnis **ZW** um eine höhere Stelle verschoben, hingegen bei einer 1 des $\mathbf{B_x}$ wird der Multiplikand **A** mit dem **ZW** addiert und das nächst Bit des **B** eingelesen und überprüft, bis das höchstwertige Bit von **B** erreicht ist und danach ausgegeben.

$$ZW <= \begin{cases} B_n = 0 : ZW = 2\,ZW \\ B_n = 1 : ZW = 2\,(ZW + A) \end{cases}$$

5. Entwurf eines Multiplizierers durch überlappendes Abtasten

Diese ist eine Abwandlung vom oberen Multiplizierer (Multiplikation durch überlappende mehrmalige Bit-Überprüfung) und wird im IBM 360/91 verwendet. Hier wird nur die Überprüfung mehrere Positionen gleichzeitig durchgeführt.

6. Kanonischer Dekodierter für die Multiplikation

In diesem Punkt wird nur die Codierung für den Kettendekoder und Zellenmultiplikation (siehe Punkt 7) erklärt.

$$D_i = B_i + C_i + 2\,C_{i+1}$$

$\mathbf{C_{i+1}=1}$, wenn von $\mathbf{B_i}$, $\mathbf{B_{i+1}}$ und $\mathbf{C_i}$ mehr als ein Bit eine EINs beinhaltet. Ist $\mathbf{D_i=-1}$, so ist die Schreibweise **/1** bzw. **!1** , hingeben bei **0** und **1** gleich. Es ergibt sich um eine Bitstelle mehr des der ausgehenden Zahl. Beim ersten Schritt ist $\mathbf{C_o=0}$.

Die Umrechnung ist simpel, wobei /1 = -1 ist, 0=0 und 1 1=1

$$D = \left(\sum_{i=0}^{n} 2^i D_i\right) + 2^{n+1} C_{n+1}$$

Bsp.: B=1111b -> D=1000/1 -> B'= 16x1 + 8x0 + 4x0 + 2x0 + 2x(-1) = 16 +(-1) = 15 Mit dieser Dekodierung kann erreicht werden, daß eine Dualzahl wenig „EINsen" enthält.

7. Kettendekodierer und Zellenmultiplizierer

Der Algorithmus ist daher einfach:

Zuerst muß der Multiplikand **A** und Multiplikator **B** in dieses in Punkt 6 erklärte Zahlensystem umgewandelt werden.

Es wird mit dem niederwertigsten Bit begonnen:

Ist $\mathbf{B_i = B_{i+1}}$, so wird das Zwischenregister *um eine Stelle höher verschoben*.
Ist $\mathbf{B_i < B_{i+1}}$, so wird zum Zwischenregister **A** *addiert* und dann *um eines Stelle höher verschoben*.
Ist $\mathbf{B_i > B_{i+1}}$, so wird zum Zwischenregister **A** *subtrahiert* und dann *um eines Stelle höher verschoben*.
Sind alle **B**-Bits und das $\mathbf{C_{n+1}}$ überprüft, so beinhaltet das Zwischenregister das Produkt aus **A x B**. Mehrere Stellen können gleichzeitig verarbeitet werden. Wenn $\mathbf{B_i=B_{i+1}}$ ist, ist diese Abarbeitung um eine Subtraktion- bzw. eine Additionszeit schneller.

Weitere Multiplizierer

8. Baukastenmultiplizierer und Mauer-Baum

Dieser Type von Multiplizierer zerlegt, $\mathbf{A=A_H\ A_L}$ und $\mathbf{B=B_H\ B_L}$, eigentlich die beiden Multiplikationsfaktoren und fügt diese durch einzelne Multiplikationen wieder zusammen. Mit dieser Option kann man erreichen, daß man einen "Groß"-Multiplizierer mit kleinen Einheiten konstruiert wird. Das Rechenprinzip ist wie folgt:

$$\mathbf{P = A \times B = (A_H\ A_L) \times (B_H\ B_L)}$$
$$\mathbf{P = A_H \times B_H + A_L \times B_H + A_L \times B_H + A_L \times B_L}$$
$$\mathbf{P = P_{HH} + P_{HL} + P_{LH} + P_{LL}}$$
$$\mathbf{P = P_{HH\text{-}H}\,(P_{HH\text{-}L}+P_{LH\text{-}H}+P_{HL\text{-}H})\,(P_{LL\text{-}H}+P_{LH\text{-}L}+P_{HL\text{-}L})\,P_{LL\text{-}L}}$$

Mit der letzten Gleichung kann man diese Additionseinheit des Multiplizierer konzipieren. $\mathbf{P_{XX\text{-}H}}$ bzw. $\mathbf{P_{XX\text{-}L}}$ sind die oberen bzw. unteren Stellen des Produktes, und wenn man diese Teilprodukte, wie oben zerlegt, addiert worden sind, so erhält man an der richtigen Position den Wert.

Da die Anordnung dieses hierarchische Systems, wie eine Mauer in einer Linienführung eines Baumes aussieht, wird als Multiplizierer MAUER-BAUM bezeichnet. Dieses System von Multiplizierer hat der Texas-Instruments-Baustein SN74S274 implementiert.

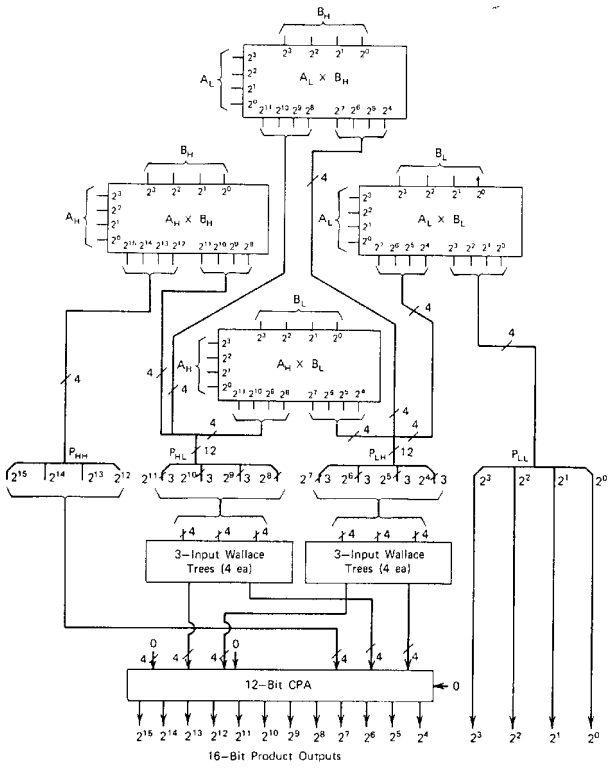

Abb.3: **Schematischer Aufbau eines Mauer-Baum-Multiplizierers**

9. direktes Zweierkomplement für Multiplizierer

Mit Hilfe des Zweierkomplements kann man negative Zahlen im binären auch darstellen. Wenn man eine Zahl im Zweierkomplement (**-B**) darstellt und eine „normale" Zahl (**A**) addiert, so erhält man die Differenz **C=A+(-B)**. Hiermit kann man 4 Typen von Volladdierern für weitere Multiplizierer konzipieren: (siehe Punkt 10)

T.0: C S = X+Y+Z ; T.1: C S = X+Y-Z ;
T.2: C S = -X-Y+Z ; T.3: C S = -X-Y-Z

Die Nomenklatur: C..Carry, S..Summe, X,Y,Z...binäre Biteingänge

10. Pezaris-Feld-Multiplizierer und Modifikationen

Der Perazis-Multiplizierer besteht aus nur Volladdierern. Die beiden zu multiplizierenden Terme $\mathbf{A} = (\mathbf{a_n})\ \mathbf{a_{n-1}}..\mathbf{a_0}$ und $\mathbf{B} = (\mathbf{b_m})\ \mathbf{b_{m-1}}..\mathbf{b_0}$ werden jeweils mit einem UND verknüpft. $\mathbf{U=a_ib_j}$, wobei $0 \leq i , j \leq n$ ist, zu beachten ist, daß $\mathbf{a_n}$ bzw. $\mathbf{b_m}$ ein Sign-Bit ist, welches Auskunft gibt, ob der Wert (bei *1)* im Zweierkomplement ist oder nicht (*0*). Die verwendet Addierer sind wie folgt aufgebaut:

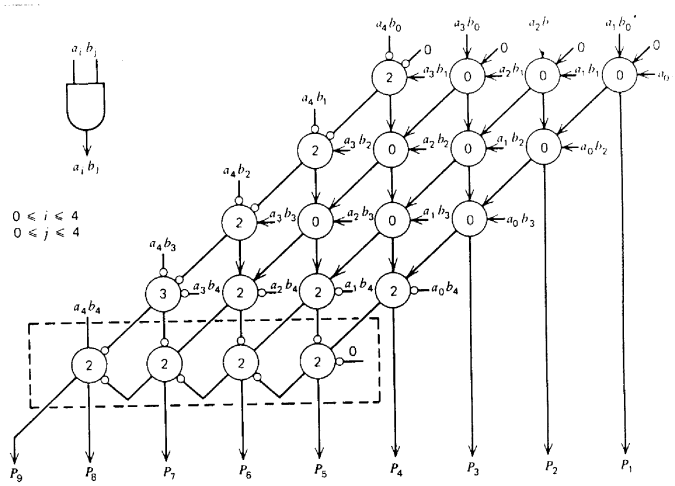

Abb.4: **Schematischer Aufbau eines Paris-Feld-Multiplizierers**

11. Baugh-Wooley-Zweikomplementenmultiplizierer

Der B-W-Zweikomponentenmultiplizierer ist schematisch dem des in Punkt 10 erklärten Multiplizierer ident. Hier werden in alle Knoten Volladdierer verwendet und die einzelnen Bits sind nach einem Schema negiert.

12. programmierbare additive multiplizierende Module (PAM)

Dieses programmierbare Modul hat 4 Eingangsvariablen (**A**, **B**, **C** & **D**) mit (je **k** Bits, $\mathbf{A=a_{k-1}..a_0}$, $\mathbf{B=b_{k-1}..b_0}$, $\mathbf{C=c_{k-1}..c_0}$, $\mathbf{D=d_{k-1}..d_0}$). Durch 2 zwei Steuerleitungen kann man 4 Modi einstellen *M00*, *M01*, *M10* und *M11*.

M_{00} (k x k): $\mathbf{P_{M=00} = A \times B + C + D}$

$$P_{M=00} = (\sum_{i=0}^{k-1} a_i 2^i)(\sum_{j=0}^{k-1} b_j 2^j) + (\sum_{r=0}^{k-1} c_i 2^i) + (\sum_{r=0}^{k-1} d_r 2^r) = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1}((a_i x b_j)2^{i+j}) + \sum_{r=0}^{k-1}(c_r + d_r)2^r$$

M_{01} (k x k):

$$P_{M=01} = (\sum_{i=0}^{k-1} a_i 2^i)(\sum_{j=0}^{k-2} b_j 2^j) + (\sum_{r=0}^{k-1} \bar{a}_i 2^i) x b_{k-1} 2^{k-1} + (\sum_{r=0}^{k-1} c_r 2^r) + (\sum_{r=0}^{k-1} d_r 2^r)$$

$$= \sum_{i=0}^{k-1}\sum_{j=0}^{k-2}((a_i x b_j)2^{i+j}) + \sum_{i=0}^{k-1}((\bar{a}_i x b_{k-1})2^{k+i-1}) + \sum_{r=0}^{k-1}(c_r + d_r)2^r$$

M_{10} (k x k):

$$P_{M=10} = (\sum_{i=0}^{k-2} a_i 2^i)(\sum_{j=0}^{k-1} b_j 2^j) + (a_{k-1} 2^{k-1})(\sum_{j=0}^{k-1} \bar{b}_j 2^j) + (\sum_{r=0}^{k-1} c_r 2^r) + (\sum_{r=0}^{k-1} d_r 2^r)$$

$$= \sum_{i=0}^{k-1}\sum_{j=0}^{k-1}[(a_i \& b_j)2^{i+j}] + \sum_{j=0}^{k-1}[(a_{k-1} \& b_j)2^{i-1}] + \sum_{r=0}^{k-1}(c_r + d_r)2^r$$

M_{11} (k x k):

$$P_{M=00} = (\sum_{i=0}^{k-2}\sum_{j=0}^{k-2} a_i b_j 2^{i+j}) + a_{k-1} b_{k-1} 2^{2k-2} + (\sum_{j=0}^{k-1} a_{k-1} \bar{b}_j 2^{k+j-1}) + (\sum_{r=0}^{k-1} a_i b_{k-1} 2^{+i-1}) + 1x2^{2k-1} +$$

$$+ (\bar{a}_{k-1} + \bar{b}_{k-1})2^{k-1} + \sum_{r=0}^{k-1}(c_r + d_r)2^r$$

13. universelles Multiplikationsnetzwerk (UMN)

Diesen Multiplizierer kann man mit dem Baukastenmultiplizierer und Mauer-Baum (siehe Punkt 8) vergleichen. Jedoch ein Knoten besteht aus einem *PAM* (siehe Punkt 12) im Modus *00*, welcher addiert und multipliziert. Für eine 4x4-Multiplikation benötigt man nur 4 *PAM*s.

14. ROM-Multiplizierer

Dieser Multiplizierer ist die einfachste Realisierung eines Multiplizierers. Hier werden die beiden Produkte der Multiplikation **A** und **B** an die Adressen (*Adr*) aufgeteilt ($\mathbf{A}$=>$Adr_0..Adr_{m-1}$, $\mathbf{B}$=>$Adr_m..Adr_{m+n-1}$), und am Datenausgang (*Dat*) der ROMs wird das Produkt ausgegeben.

15. logarithmischer Multiplizierer

Hier wird die Multiplikation von den logarithmierten Werten von A und B addiert und delogarithmiert. Die Logarithmierung und die Delogarithmierung stellt eine gewisse Problematik dar, da diese auf Rundungsfehler aufgebaut ist und das Ergebnis nicht ganz genau stimmt.

$$\mathbf{P = A \times B = \exp[\ \ln(A) + \ln(B)\ ]}$$

16. digitale-analoge Multiplizierer

Mit Hilfe von je einen DA-Wandler (Digital-Analog-Umsetzer) werden die beiden zu multiplizierenden digitale Signale in ein analoges Signal umgewandelt. Mit einem analogen Multiplizierer, welche meistens aus 2 Logarithmierern, einem Addierer und einem Entlogarithmierer besteht (siehe Punkt 15), das Produkt gebildet und mittels einen AD-Wandler (Analog-Digital-Wandler) in ein digitales Signal konvertiert.

Literaturquelle

Die Hauptinformationsquelle zu diesem Artikel war das Referat **MULTIPLIZIERER** geschrieben im Dezember 1995/Jänner 1996.

Zusammengefaßte Kapiteln von

COMPUTER ARITHMETIC Principles Architectures and Design
Kai Hwang
Chapter 5 & 6,

Als ergänzende Literatur sind folgende Artikeln zu erwähnen
IEEE Transactions on Computer Vol. 43 No. 1, Jan. 1994
High-Speed Area-Effecient Multiplier Desin Using Multiple-Vauled Current-Mode Circuits

IEEE Transaction on Computers Vol. 43 No. 3, Mar. 1994
Fast Hardware-Based Algorithmus for Elementary Computations Using Rectangular Multiplier

IEEE Transcations on Computers Vol. 42 No. 3, Mar. 1993 **Systolic Modular Multiplication**

□