Dokumentation zur GFS „Digitales Rechenwerk“

Dominik Eisele

24. Juni 2015

Inhaltsverzeichnis

1 Rechnen mit dualen Zahlen **2**

1.1 Addition von Dualzahlen 2

1.2 Subtraktion von Dualzahlen 3

1.3 Subtraktion mit dem Zweierkomplement 4

1.4 Multiplikation von Dualzahlen 6

2 Umsetzung in eine Digitale Rechenschaltung **7**

2.1 Addierer 7

2.1.1 Halbaddierer 7

2.1.2 Volladdierer 8

2.2 Subtrahierer 9

2.3 Addierwerk 10

2.4 Multiplizierer 12

2.5 BCD-Eingabe 14

2.6 Umwandlung 15

2.7 7-Segment Anzeige 17

1 Rechnen mit dualen Zahlen

1.1 Addition von Dualzahlen

Die Addition von Dualzahlen verläuft im Prinzip wie die Addition von Dezimalzahlen, sobald die Addition an einem Stellenwert den maximalen Stellenwert übersteigt erfolgt ein Übertrag auf die nächste Stelle.

Rechenregeln:

$$\begin{aligned} 0+0 &= 0 \\ 0+1 &= 1 \\ 1+1 &= 0\ (+\ 1\ \text{Übertrag}) \\ 1+1+1 &= 1\ (+\ 1\ \text{Übertrag}) \end{aligned}$$

Beispiel:

$$\begin{array}{lrcl} & 111010010 & = & 466 \\ + & 001110100 & = & 116 \\ \text{Ü} & 111111 & & \\ \hline & 1001000110 & = & 582 \end{array}$$

1.2 Subtraktion von Dualzahlen

Bei der Subtraktion von Dualzahlen gelten die gleichen Rechenregeln, wie bei der Subtraktion von Dezimalzahlen.

Rechenregeln:

$$
\begin{aligned}
0-0 &= 0\\
0-1 &= 1\ (+\ 1\ \text{Entlehnung})\\
1-0 &= 1\\
1-1 &= 0\\
0-1-1 &= 0\ (+\ 1\ \text{Entlehnung})\\
1-1-1 &= 1\ (+\ 1\ \text{Entlehnung})
\end{aligned}
$$

Beispiel:

$$
\begin{array}{lrcl}
 & 111010010 & = & 466\\
- & 001110100 & = & 116\\
\text{E} & 11111 & & \\
\multicolumn{4}{l}{\text{———————}}\\
 & 101011110 & = & 350
\end{array}
$$

1.3 Subtraktion mit dem Zweierkomplement

Da, in der Digitaltechnik, für die Subtraktion von Dualzahlen keine logische Verknüpfung existiert, ist man gezwungen eine Subtraktion in eine Addition umwandeln.

$$\begin{aligned} 2-6 &= (-4) \\ 2+(-6) &= (-4) \end{aligned}$$

Diese Umwandlung geschieht mit Hilfe des Zweierkomplements. Es wird gebildete in dem man den Subtrahend auf die volle Stellenzahl erweitert (Nullen nach links auffüllen). Hierbei muss die Breite der Komplementdarstellung beider Zahlen berücksichtigt werden. Üblich sind 4, 8, 16, 32 und 64 Bit. Als nächstes muss der Subtrahend negiert werden, das heißt man lässt jedes einzelne Bit kippen. Zu dem gebildeten bitweisen Komplement muss +1 hinzuaddiert werden. Das nun erhaltene Zweierkomplement muss man mit dem Minuenden addieren um das Zweierkomplement-Ergebnis der eigentlichen Subtraktion zu erhalten. Um das nun eigentliche Ergebnis zu erhalten muss man das Ergebnis negieren und wieder +1 addieren. Das höchstwertigste Bit des Zweierkomplement-Ergebnisses, stellt dabei das Vorzeichen da, und wird nicht zur Negation verwendet. Wenn es eine 1 ist, ist das Endergebnis negativ, bei einer 0 ist es positiv.

Beispiel:

$2 - 6 = ?$

1. Schritt: In eine Dualzahl wandeln:

$2 - 6 \Rightarrow 10 - 110$

2. Schritt: Stellen auffüllen:

$0010 - 0110 = ?$

3. Schritt: Bits negieren:

$0110 \Rightarrow 1001$

4. Schritt: Hinzuaddieren von 1:

$1001 + 0001 = 1010$

5. Schritt: Minuend und Zweierkomplement addieren:

$0010 + 1010 = 1100$

6. Schritt: Ergebnis negieren:

$100 \Rightarrow 011$

7. Schritt: Hinzuaddieren von 1:

$011 + 001 = 100$

8. Schritt: In eine Dezimalzahl wandeln:

$100 \Rightarrow 4$; da das höchstwertige Bit 1 ist: Endergebnis $= -4$

1.4 Multiplikation von Dualzahlen

Auch die binäre Multiplikation wird nach denselben Regeln durchgeführt, wie die dezimale Multiplikation. Dabei werden Produkte mit den einzelnen Stellen des Multiplikators gebildet und anschließend Stellenrichtig addiert. Gegenüber der dezimalen Multiplikation bringt die binäre Multiplikation Vereinfachungen mit sich. Da die Stellen des Multiplikators nur die Zahlenwerte Null und Eins annehmen können, muss der Multiplikand nur mit Null und Eins multipliziert werden. Bei der binären Multiplikaton kommen als Teiloperationen nur die Addition und Verschiebung vor.

Beispiel:

$$
\begin{array}{rr}
 & \underline{1011 \times 1010} \\
 & 1011 \\
 & 0000 \\
 & 1011 \\
+ & 0000 \\
\hline
 & 1101110
\end{array}
$$

2 Umsetzung in eine Digitale Rechenschaltung

2.1 Addierer

2.1.1 Halbaddierer

Ein Halbaddierer besitzt zwei Ein-, und zwei Ausgänge. An die Eingänge x und y werden jeweils die Ziffern angelegt die man addieren möchte. An dem ersten Ausgang liegt die Summe s der Addition an, am zweiten Ausgang der Übertrag c.
Daraus ergibt sich die Wertetabelle 1.

x	**y**	**Übertrag c**	**Summe s**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabelle 1: Wertetabelle Halbddierer

In Schaltungen wird der Halbaddierer aus zwei Bauteilen zusammengesetzt, ein Exklusiv-ODER (XOR) und ein UND (AND).
Der Aufbau des Halbaddierers ist in Abbildung 1 dargestellt.

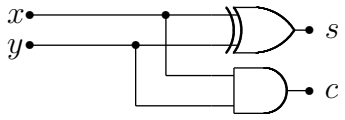

Abbildung 1: Halbaddierer

2.1.2 Volladdierer

Der Volladdierer besteht aus zwei Halbaddierern und einem ODER. Da ein Volladdierer einen zusätzlichen Eingang (c_{in}) hat, kann man mit ihm den Übertrag aus einer vohergegangenen Addition mit in die Rechnung einbeziehen. Man kann somit mehrere Volladdierer hintereinander schalten um größere Zahlen miteinander zu addieren. Dabei verbindet man den Carry out Ausgang mit dem Carry in Ausgang des höherwertigen Volladierers.
Daraus ergibt sich die Wertetabelle 2.

x	**y**	$\mathbf{c_{in}}$	$\mathbf{c_{out}}$	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabelle 2: Wertetabelle Volladdierer

In Schaltungen wird der Volladdierer aus zwei Halbaddierern und einem ODER (OR) zusammengesetzt.
Der Aufbau des Halbaddierers ist in Abbildung 2 dargestellt.

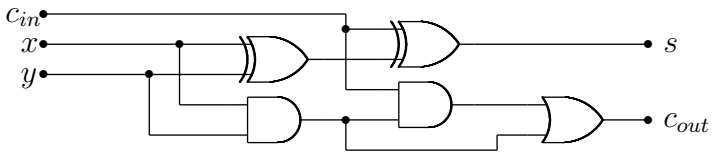

Abbildung 2: Volladdierer

2.2 Subtrahierer

Beim Subtrahierer wird der Volladdierer durch einen Steuereingang erweitert. An diesem legt man für eine Addition eine 0, und für eine Subtraktion eine 1 an. Diesen Steuereingang legt man, zusammen mit dem y-Eingang, an ein Exklusiv-ODER (XOR). Der Ausgang dieses Exklusiv-ODERs wird an den ursprünglichen y Eingang angelegt. Somit wird der y Wert, bei einer am Steuereingang anliegenden logischen 1 negiert. Bei dem niederwertigsten Subtrahierer muss zudem das Steuersignal auf den Carry in Eingang gelegt werden um die, für die Zweierkomplementrechnung benötigte, 1 zu addieren. Zu sehen ist dieser Aufbau in Abbildung 3.

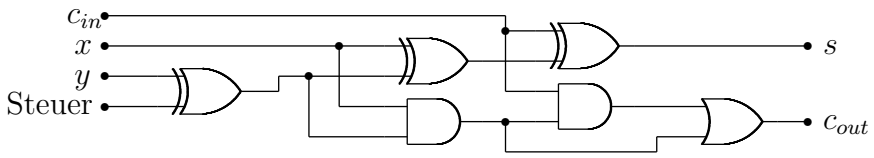

Abbildung 3: Volladdierer mit Steuerleitung zum Subtrahieren

2.3 Addierwerk

Das in Abbildung 4 gezeigte Schaltnetz besteht aus acht kaskadierten Volladdierern. Zusätzlich besitzen sie jeweils einen Steuereingang, mit welchem der Summand, der aus den y Bestandteilen aufgebaut ist, negiert werden kann, sodass mit Schaltnetz auch Subtraktionen durchgeführt werden können.

Um eine Rechenoperation durchzuführen muss man die Summanden x und y an die Eingänge x_1 bis x_n und y_1 bis y_n anlegen. An die Steuereingänge legt man eine logische 0 für eine Addition, und eine logische 1 für eine Subtraktion an. Die Endsumme liegt an den Ausgängen z_1 bis z_n an.

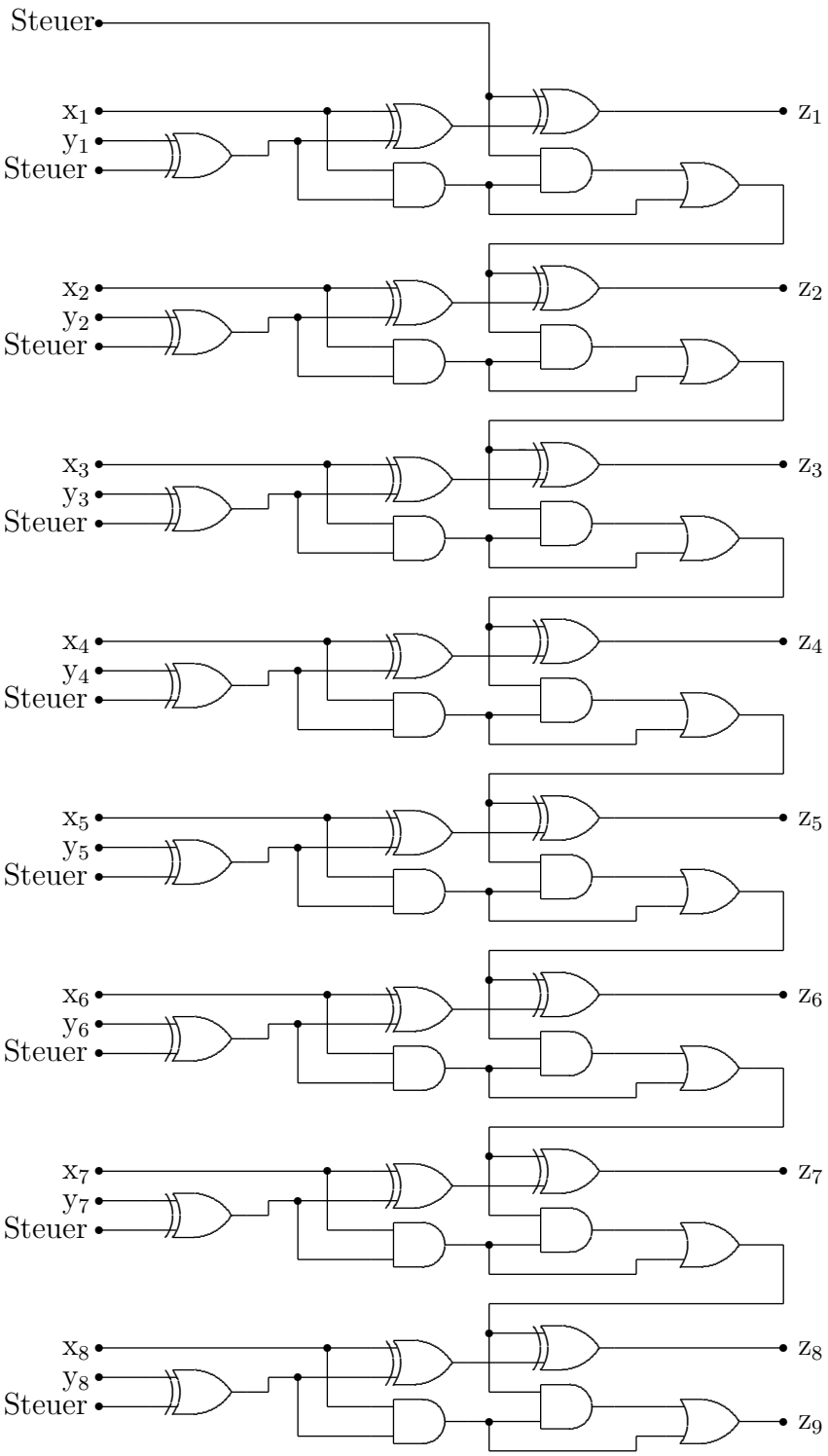

Abbildung 4: Steuerbares Addierwerk

2.4 Multiplizierer

Der in Abbildung 6 gezeigte Multiplizierer kann zwei acht Bit Zahlen miteinander multiplizieren. Dazu legt man seine erste Zahl ($2.1 - 2.8$) an ein Schieberegister an, sodass die Zahl bei jedem Takt um eine Stelle verschoben werden kann. Dann wird pro Takt die höchstwertigste Ziffer des Multiplikators abgegriffen, die anschließend mit dem Multiplikant multipliziert wird. Da die abgegriffene Ziffer nur die Zustände „Null“ und „Eins“ annehmen kann, kann man die Multiplikation mit einfachen UND-Verknüpfungen realisieren. Um die Produkte auch Stellenrichtig addieren zu können muss man die Zwischenergebnisse ebenfalls pro Takt verschieben. Dies wird gelößt indem man ein Register nimmt, welches das Ergebniss um eine Stelle versetzt wieder auf den Addierer gibt. Der selben Aufbau aus Register und Addierer wird auch verwendet um die, aus dem ersten Addierer hinausgeschobenen, Stellen aufzufangen. Das Produkt kann man, nach acht Takten Rechenzeit, an den Ausgängen der Addierer abgreifen. Da die Steuereinheit aus Abbildung 5 das Taktsignal einen Takt zu spät stoppt, muss man das Ergebnis durch zwei teilen, indem man die Ziffern des Produktes jeweils an dem nächst höherwertigen Pin abgreift.

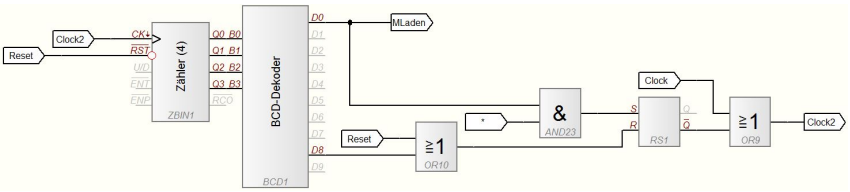

Abbildung 5: Steuereinheit 8-Bit Multiplizierer

Die in Abbildung 5 sichtbare Steuereinheit des 8-Bit Multiplizierers ist dafür verantwortlich, dass der Takt „ClMult“ am Anfang er Rechenoperation gestartet, und nach acht Takten wieder gestoppt wird. Wenn D_0 nach einem Reset 1 ist, wird das ladbare Schieberegister geladen. Ist zusätzlich dazu noch der Taster „* “ gedrückt, wird der Ausgang $\bar{Q}$ des Flip-Flops 0, sodass der Main-Clock auch der Clock des Multiplizierers wird. Acht

Takte später, wenn der Ausgang D_8 1 ist, wird der Reseteingang des Flip-Flops 1, sodass $\bar{Q}$ wieder 1 ist und der Takt blockiert wird. So wird der Multiplizierer gestoppt, sodass das Produkt an den Ausgängen ErBin1 bis ErBin14 abgelesen werden kann.

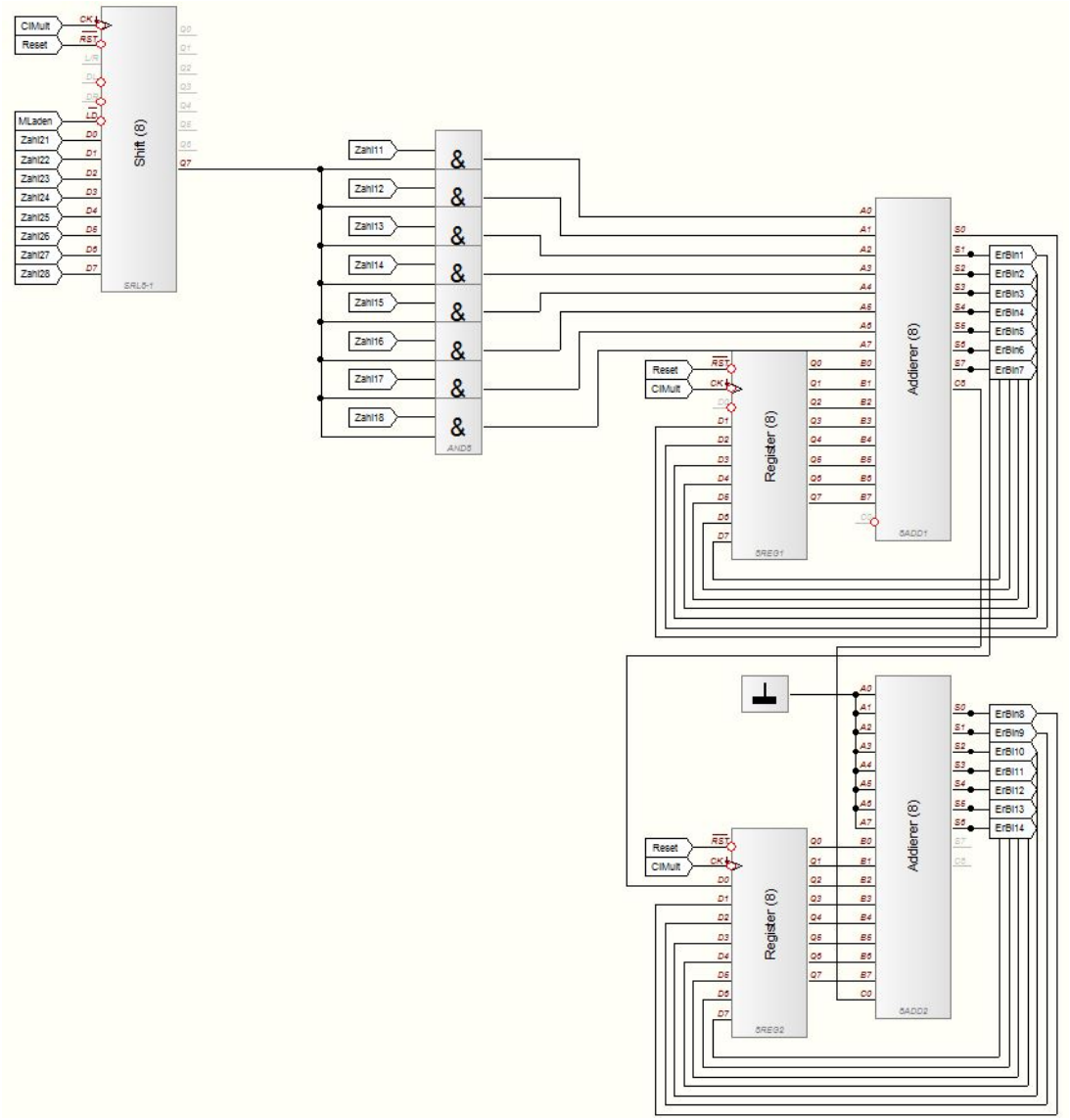

Abbildung 6: 8-Bit Multiplizierer

2.5 BCD-Eingabe

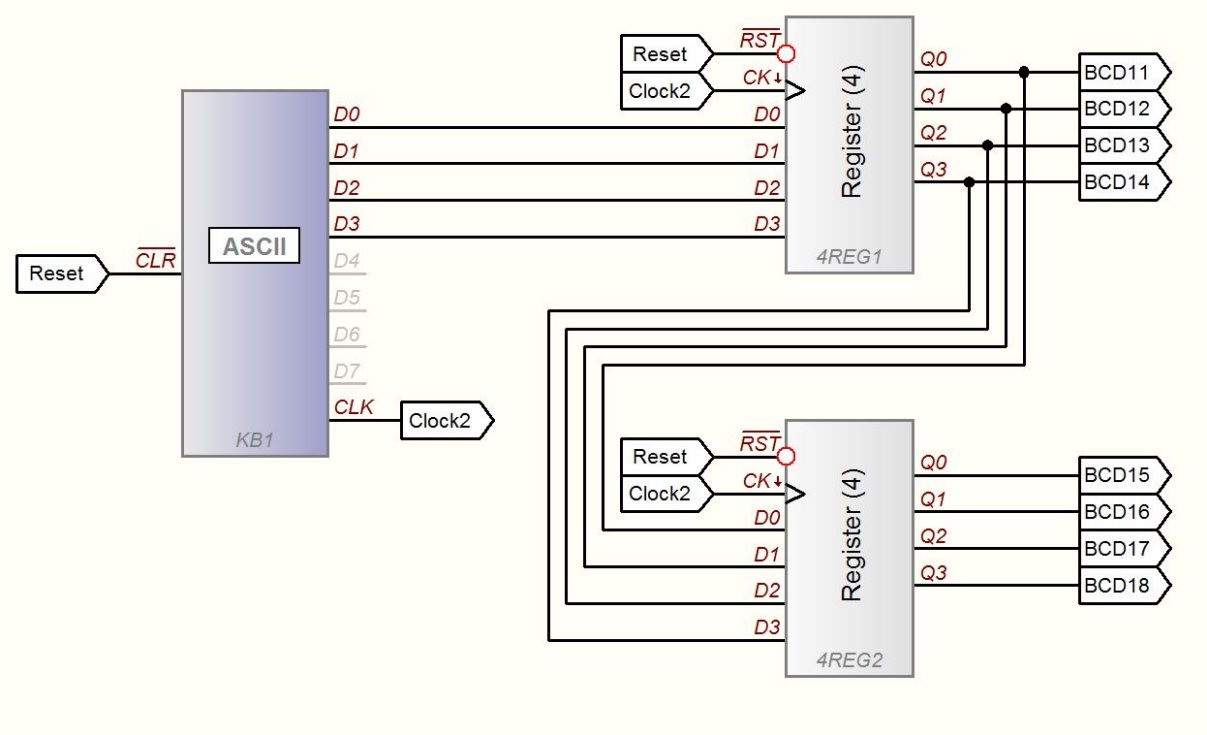

Abbildung 7: BCD-Eingabe

Die digitale Eingabe des Rechenwerks erfolgt über ein ASCII-Eingabe Feld. Dabei liegt an den Ausgängen D_0 bis D_3 die zuletzt eingegebene Ziffer im Binärcode an da, durch das weglassen der oberen vier Bit im ASCII-Code die Ziffern null bis neun automatisch im BCD-Code anliegen. Der Ausgang CLK ist dann HIGH solange eine Taste gedrückt wird. Dieser CLK Ausgang wird auf den Clock-Eingang der zwei nachfolgenden Register gelegt. Sobald eine Taste gedrückt wurde, wird die zuletzt eingegebene Ziffer, mit fallender Flanke, in das erste Register übernommen. Folgt ein weiterer Tastendruck wird, wieder mit fallender Flanke, die erste Ziffer in das zweite Register übernommen, die niederwertigere Ziffer in das erste Register übernommen. Die eingegebene Zahl kann man dann an den Registerausgängen BCD1.1 bis BCD1.8 ablesen. Um zwei Operanden eingeben zu können, muss die Schaltung aus Abbildung 7 doppelt vorhanden sein.

2.6 Umwandlung

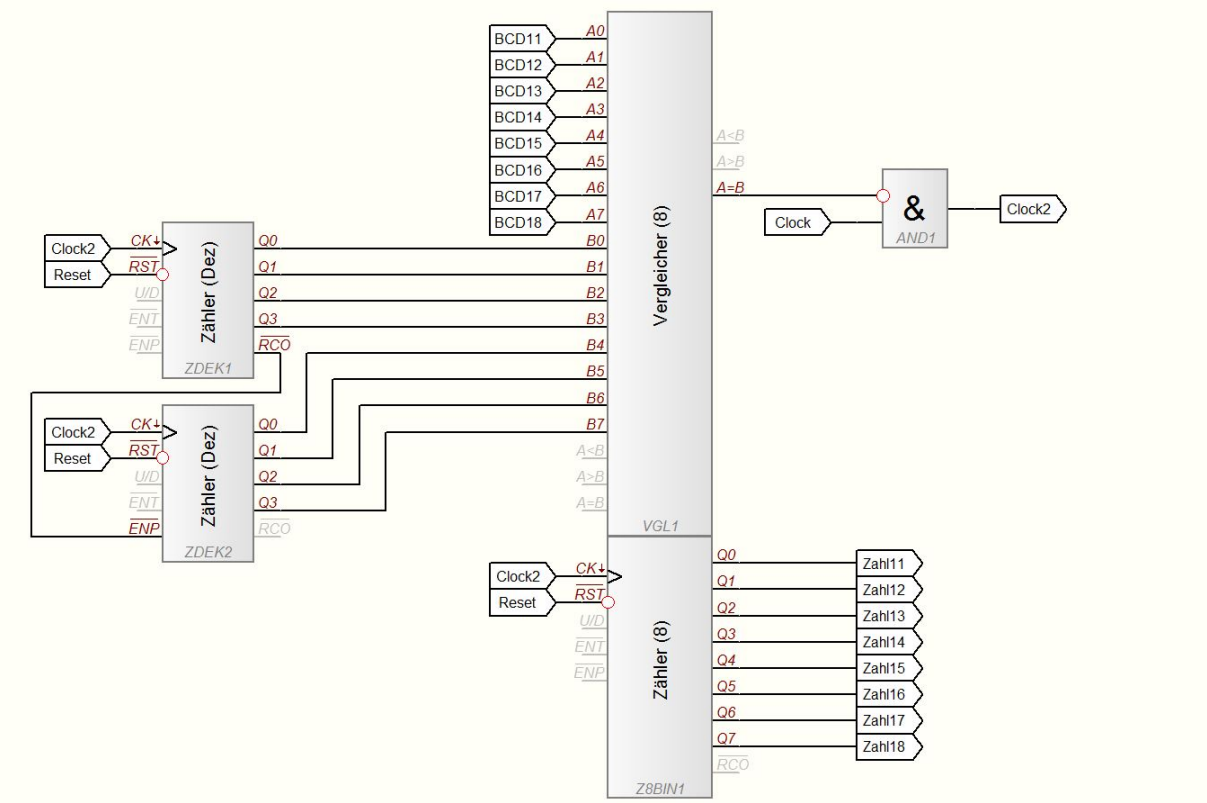

Abbildung 8: Wandler von BCD zu Binär

Da das Rechenwerk mit binären Zahlen rechnet muss man die eingegebenen BCD-Zahlen in Binär-Zahlen wandeln. Da die heutzutage üblicherweise Verwendete Konvertierung sehr komplex ist, habe ich für meine GFS eine simplere Methode gewählt. Diese Variante ist nur Effektiv bei kleineren Zahlenbereichen, da die Anzahl benötigter Takte genau so hoch ist, wie die Zahl selber. Bei einer Frequnez von, z.B 500Hz braucht die Wandlung der Zahl 100 nur $0.2s$, die Wandlung der Zahl 10.000 hingegen schon $20s$. Bei dem Rechenbeispiel mit den höchstmöglichsten Operanden ($99 \cdot 99 = 9.801$) braucht die eigentliche Rechnung nur 8 Takte, die Wandlung hingegen 9.801 Takte. Bei einem angelegten Takt von 500Hz beträgt die eigentliche Rechenzeit, der Multiplikation, $0{,}016s$, die Wandlung hingegen benötigt $19.602s$.

Für die Wandlung werden zwei Zähler und ein Vergleicher benötigt. Der erste Zähler ist

dabei ein BCD-Zähler, der zweite ein Binärzähler. Dabei ist es wichtig dass die beiden Zähler im gleichen Takt zählen. Der Vergleicher vergleicht das Eingabesignal mit dem Ausgang des BCD-Zählers. Sind diese beiden Signale identisch so wird der Takt unterbrochen und somit werden beide Zähler gestoppt. Da sie im Gleichen Takt zählen, und auch den selben Reset besitzen, kann man nun das Ergebniss binär an den Ausgängen des zweiten Zählers ablesen.

Für die Wandlung von binär zu BCD kann man das selbe Prinzip anwenden, man muss allerdings die beiden Zähler tauschen, sodass das binär-Ergebniss mit dem binär-Zähler verglichen wird, und man dass Endergebniss an den Ausgängen des Dezimalzählers ablesen kann.

2.7 7-Segment Anzeige

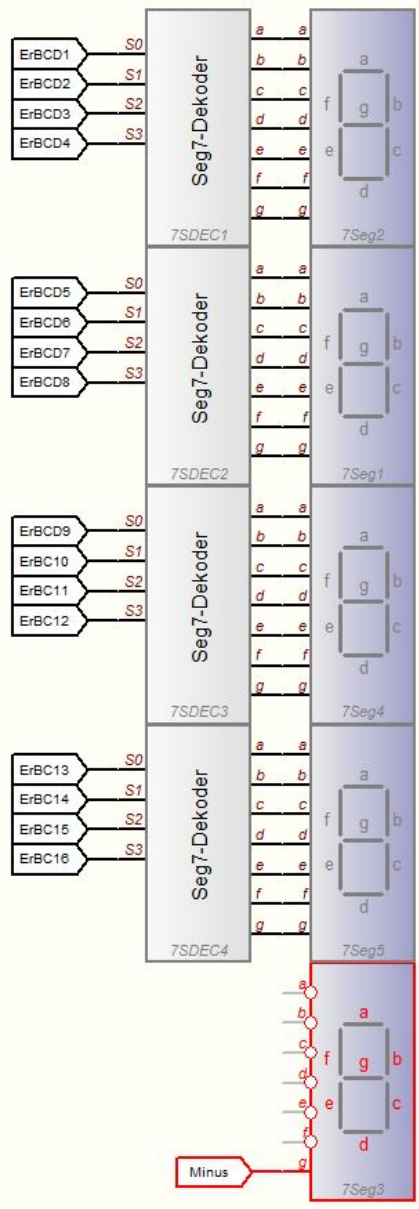

Abbildung 9: 7-Segment Anzeige

Um das Endergebniss anzuzeigen wird es, BCD-Codiert, Ziffer für Ziffer auf die Eingänge eines 7-Segment-Wandlers gelegt, seine Ausgänge gehen wiederum auf jeweils ein 7-Segment Anzeigeelement.