

Kapitel 6

Theorie 600 Halbaddierer und Volladdierer

Der bürgerliche Algorithmus des schriftlichen Addierens zerlegt die binäre Addition in die folgenden elementaren Additionen:

A + B = S, Übertrag C

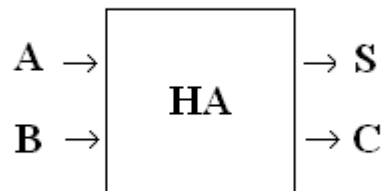
0 + 0 = 0, Übertrag 0

0 + 1 = 1, Übertrag 0

1 + 0 = 1, Übertrag 0

1 + 1 = 0, Übertrag 1

Eine Digitalschaltung, die diese Aufgaben rechnen soll, habe die Eingänge A und B und die Ausgänge S (sum) und C (carry):



Die entsprechende Funktionstabelle sieht folgendermaßen aus:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Diese Schaltung wird als **Halbaddierer** bezeichnet. Aus der Funktionstabelle liest man nach dem bekannten Verfahren die allgemeine disjunktive Normalform der Booleschen Funktion ab:

(I) $S = (A \wedge \neg B) \vee (\neg A \wedge B)$

(II) $C = A \wedge B$

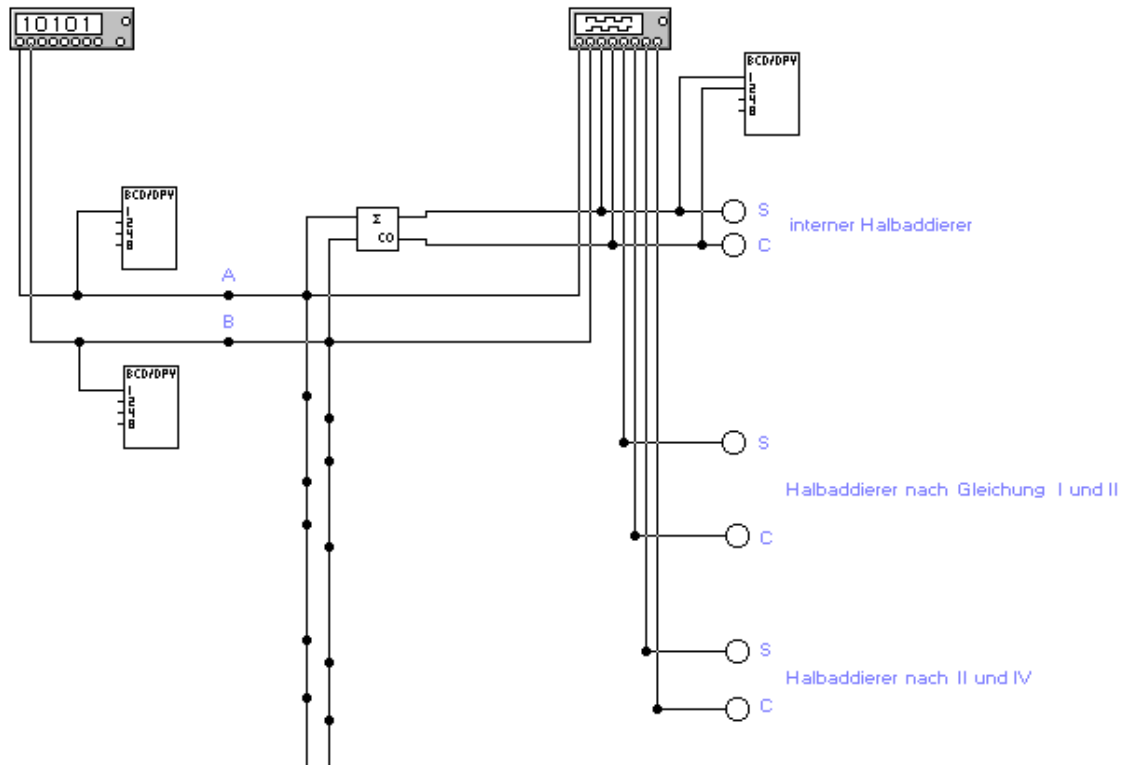
Die Gleichung (I) entspricht der XOR-Funktion. Der Halbaddierer läßt sich also durch die beiden folgenden Gleichungen beschreiben:

(III) $S = A \text{ XOR } B$

(IV) $C = A \wedge B$

Versuch 605 Halbaddierer

In der Datei v605 finden Sie folgende Meßschaltung:

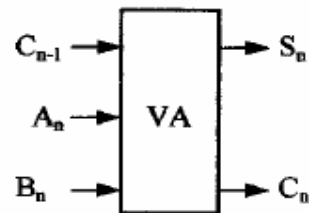


- Bauen Sie in diese Schaltung einen Halbaddierer nach den Gleichungen (I) und (II) und einen Halbaddierer nach den Gleichungen (III) und (IV).
- Lassen Sie mit dem Logik-Analysator für die Halbaddierer Oszillogramme bei 1 KHz Bitmuster-Generator-Frequenz erzeugen.

Der Halbaddierer kann bei der Addition zweier mehrstelliger Binärzahlen die Addition einer Stelle durchführen, wenn von der Addition in vorhergehenden Stellen (rechte Nachbarstelle) kein Übertrag entstanden ist. Nur in der ersten Stelle (LSB) gibt es keinen Übertrag. In allen anderen Stellen müssen drei Binärziffern addiert werden: der Übertrag von der vorhergehenden Stelle und die beiden Summanden. Dafür braucht man eine Digitalschaltung mit drei Eingängen und zwei Ausgängen, die die folgenden elementaren Additionen berechnen kann:

$A_n + B_n + C_{n-1} = S_n$	Übertrag: C_n
0 + 0 + 0 = 0	Übertrag: 0
0 + 0 + 1 = 1	Übertrag: 0
0 + 1 + 0 = 1	Übertrag: 0
0 + 1 + 1 = 0	Übertrag: 1
1 + 0 + 0 = 1	Übertrag: 0
1 + 0 + 1 = 0	Übertrag: 1
1 + 1 + 0 = 0	Übertrag: 1
1 + 1 + 1 = 1	Übertrag: 1

Darin bedeuten die Indizes, dass die Schaltung bei der Addition zweier mehrstelliger Binärzahlen die Addition für die n-te Stelle durchführen soll.



Die Schaltung heisst Volladdierer. Aus ihrer Funktionstabelle liest man die folgende disjunktive Normalform für S_n ab: und

$$(V) \quad (A_n \wedge \neg B_n \wedge \neg C_{n-1}) \vee (\neg A_n \wedge B_n \wedge \neg C_{n-1}) \vee (\neg A_n \wedge \neg B_n \wedge C_{n-1}) \vee (A_n \wedge B_n \wedge C_{n-1})$$

Daraus kann man

$$(VI) \quad S_n = A_n \text{ XOR } B_n \text{ XOR } C_{n-1}$$

ableiten. Für C_n ergibt die Funktionstabelle die disjunktive Normalform

$$(VII) \quad C_n = (\neg A_n \wedge B_n \wedge C_{n-1}) \vee (A_n \wedge \neg B_n \wedge C_{n-1}) \vee (A_n \wedge B_n \wedge \neg C_{n-1}) \vee (A_n \wedge B_n \wedge C_{n-1})$$

Daraus kann man die folgenden drei Varianten ableiten:

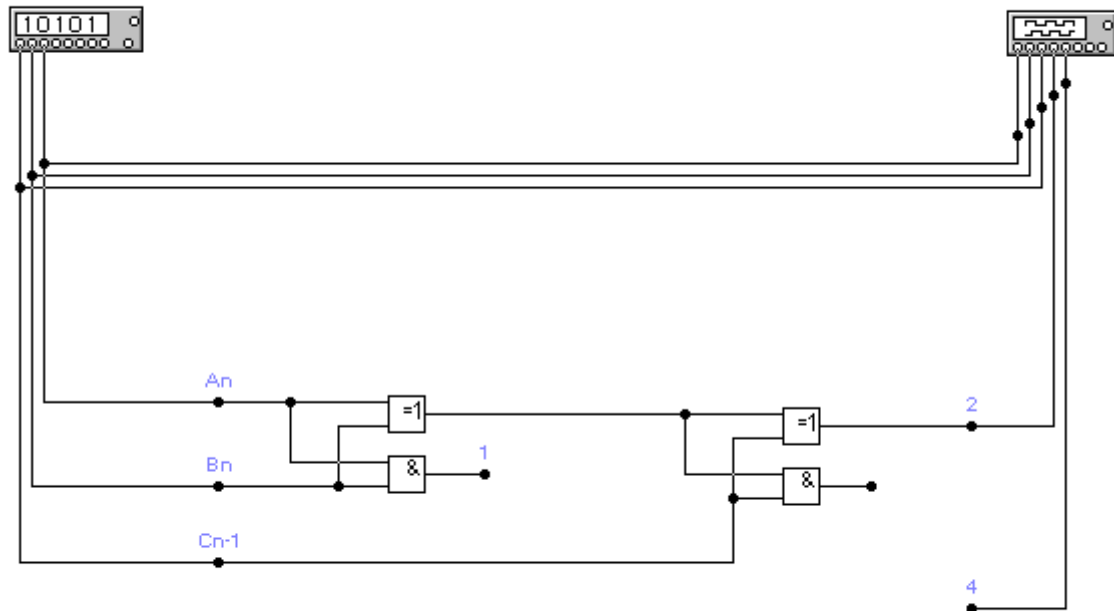
$$(VIII) \quad C_n = (A_n \wedge B_n) \vee ((A_n \text{ XOR } B_n) \wedge C_{n-1})$$

$$(IX) \quad C_n = (A_n \wedge B_n) \vee (A_n \wedge C_{n-1}) \vee (B_n \wedge C_{n-1})$$

$$(X) \quad C_n = (A_n \wedge B_n) \vee (A_n \vee B_n) \wedge C_{n-1}$$

Versuch 615 Volladdierer

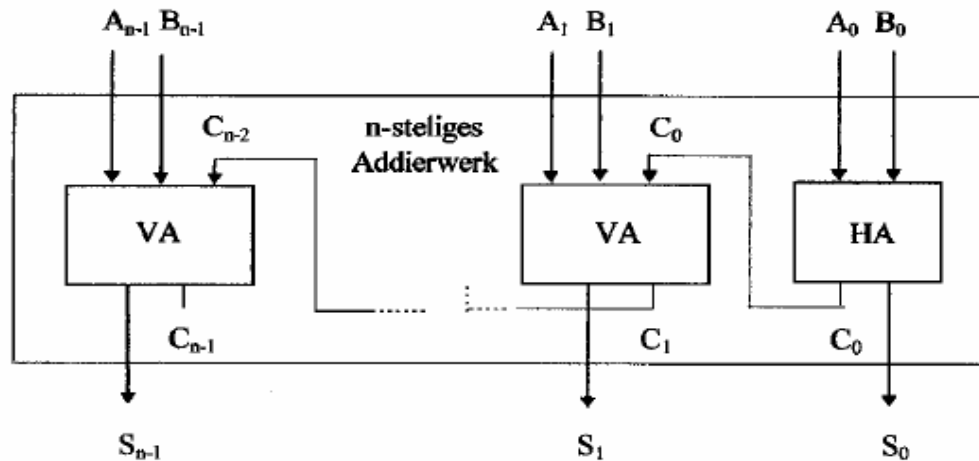
In der Datei v615 finden Sie zwei hintereinandergeschaltete Halbaddierer:



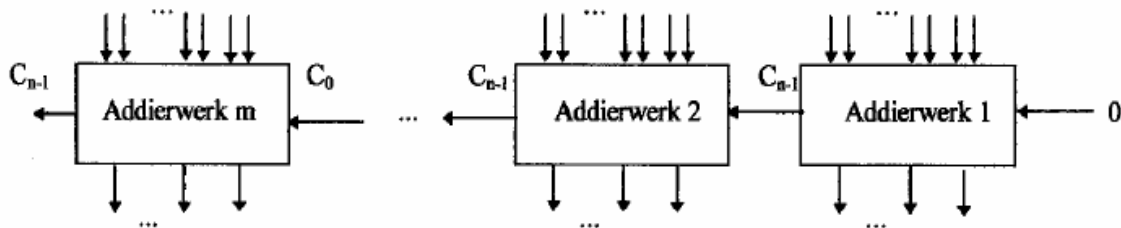
Ergänzen Sie die Schaltung mit Hilfe der beiden Gleichungen (VI) und (VIII) so, dass ein Volladdierer entsteht.

Versuch 620 Ripple-Carry-Adder

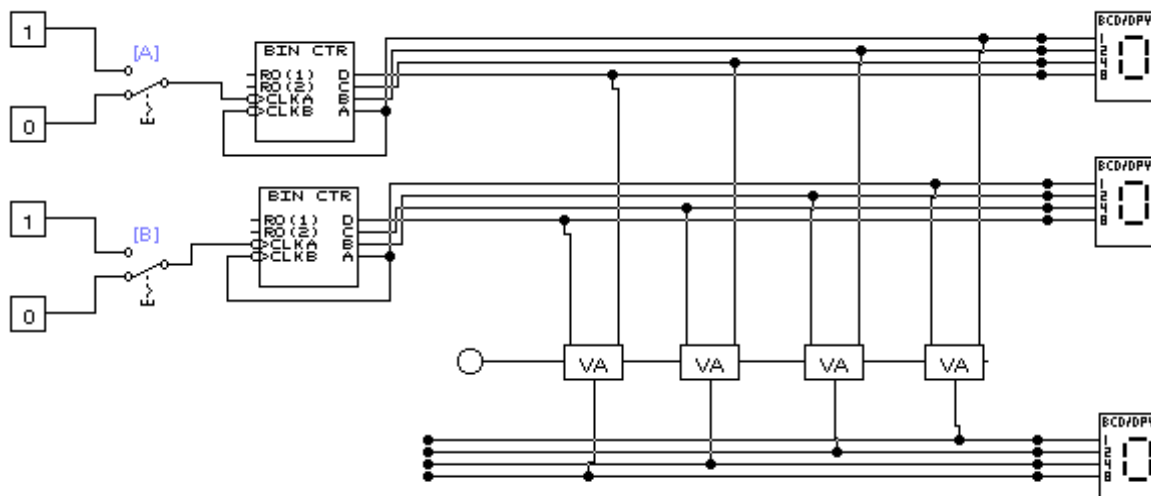
Mit $n-1$ Volladdierern und einem Halbaddierer kann man eine Digitalschaltung aufbauen, die zwei n -stellige Binärzahlen $A_{n-1} \dots A_0$ und $B_{n-1} \dots B_0$ addiert:



Wenn man den Übertrag der höchsten Stelle (C_{n-1}) dieses Ripple-Carry-Adders als zusätzlichen Ausgang herausführt und an Stelle des Halbaddierers in der letzten Stelle einen Volladdierer einbaut, dessen C-Eingang C_{0-1} zusätzlicher Eingang wird, erhält man ein kaskadierbares Addierwerk, mit dem man $(m \cdot n)$ -stellige Addierwerke bauen kann:



In der Datei v620 finden Sie die folgende Testschaltung:

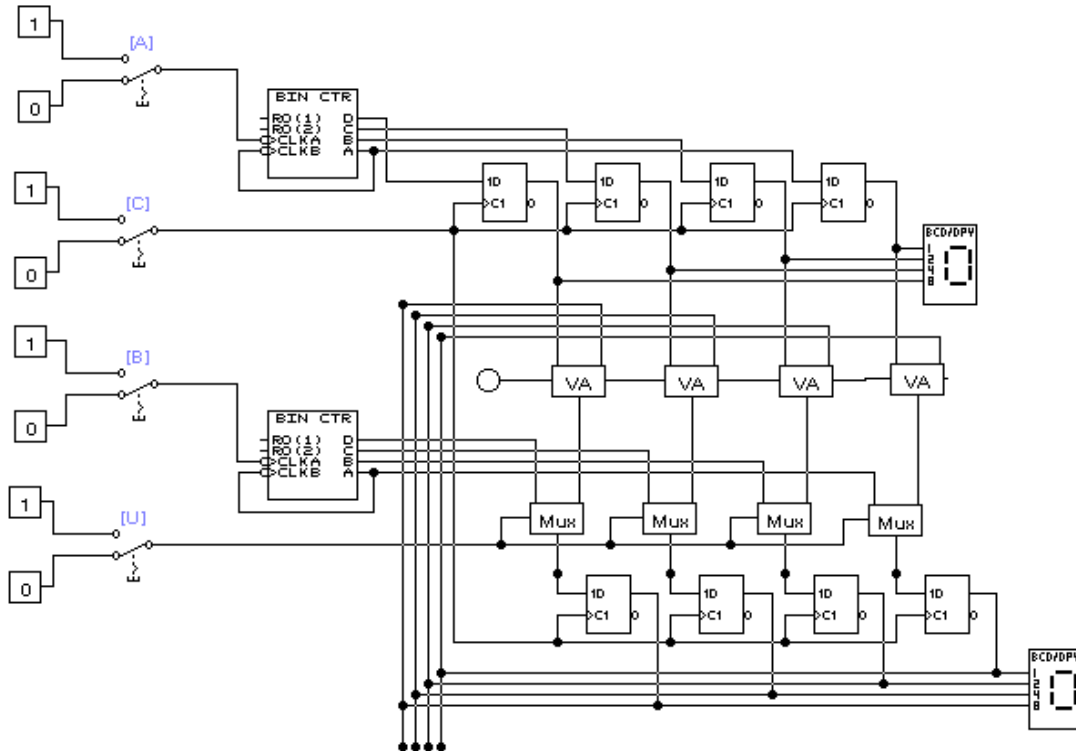


Das Makro VA ist leer. Bauen Sie in dieses Makro Ihren Volladdierer aus dem vorherigen Versuch ein. Testen Sie das entstandene Addierwerk.

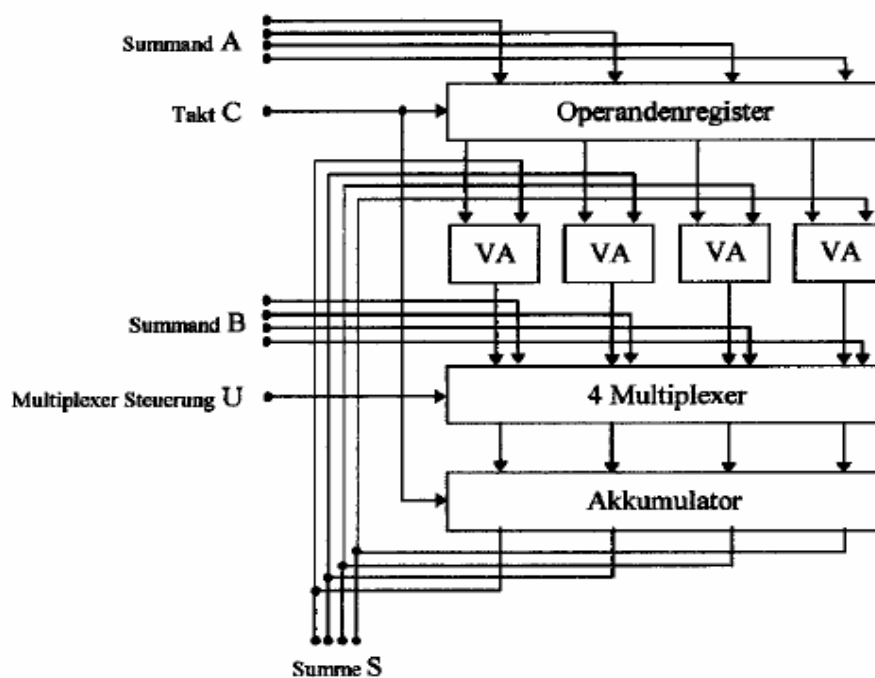
Versuch 625 Synchrones Addierwerk

Das Addierwerk aus Versuch 620 arbeitet asynchron, d.h. man kann zu jedem beliebigen Zeitpunkt an den Eingängen der Volladdierer Summanden eingeben. Die Schaltung addiert allerdings nur dann fehlerlos, wenn man ihr genügend Zeit für die Erzeugung der Summe lässt.

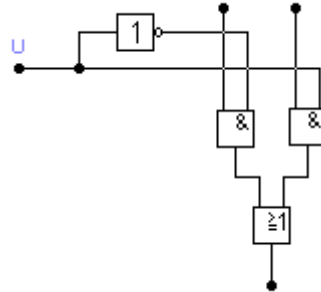
In der Datei v625 finden Sie ein synchrones Addierwerk:



Dieser Schaltung liegt das folgende Schema zu Grunde:



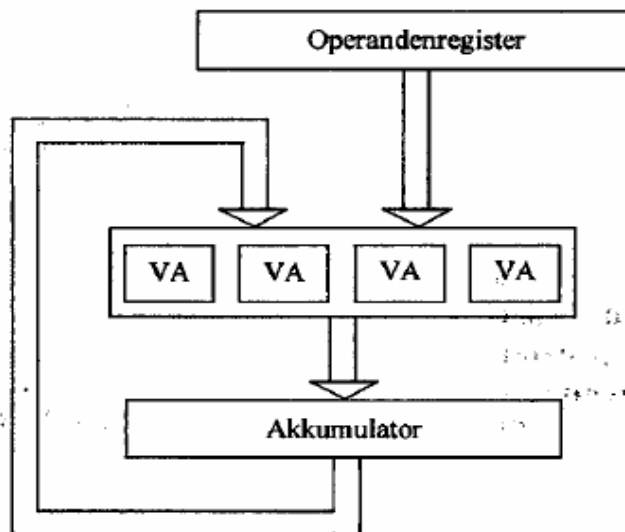
Das im vorherigen Versuch entwickelte asynchrone Addierwerk mit vier Volladdierern (d.h. eine kombinatorische Schaltung) ist der Kern dieser Schaltung und wird durch Hinzuschalten der beiden Register Akkumulator und Operandenregister zur sequentiellen Schaltung. Die Multiplexer-Blackbox enthält vier Multiplexer des folgenden Typs.



Mit dem Steuereingang U können die Eingänge des Akkumulators entweder mit den Ausgängen der Volladdierer oder mit den Schaltungseingängen B verbunden werden, an denen der Summand B eingegeben werden kann. Der Additionsprozess läuft synchron mit dem Takt in folgenden Schritten ab:

1. Zeitpunkt t_0 : Summanden A und B an den Eingängen A und B eingeben und ab einem Zeitpunkt t_0 konstant halten.
2. Zeitpunkt t_1 : Summand B mit dem Multiplexer auf die Eingänge des Akkumulators schalten.
3. Zeitpunkt t_2 : Summanden A und B mit steigender Taktflanke in das Operandenregister bzw. den Akku eingeben.
4. Zeitpunkt t_3 : Multiplexer umschalten. Dann steht die Summe $A + B$ an den Ausgängen der Multiplexer.
5. Zeitpunkt t_2 : An den Ausgängen der Multiplexer stehende Summe $A+B$ mit steigender Taktflanke in den Akkumulator takten.
6. Ein neuer Zyklus kann beginnen.

Die Grundidee der Schaltung kommt auch in der folgenden abstrakteren Darstellung zum Ausdruck:

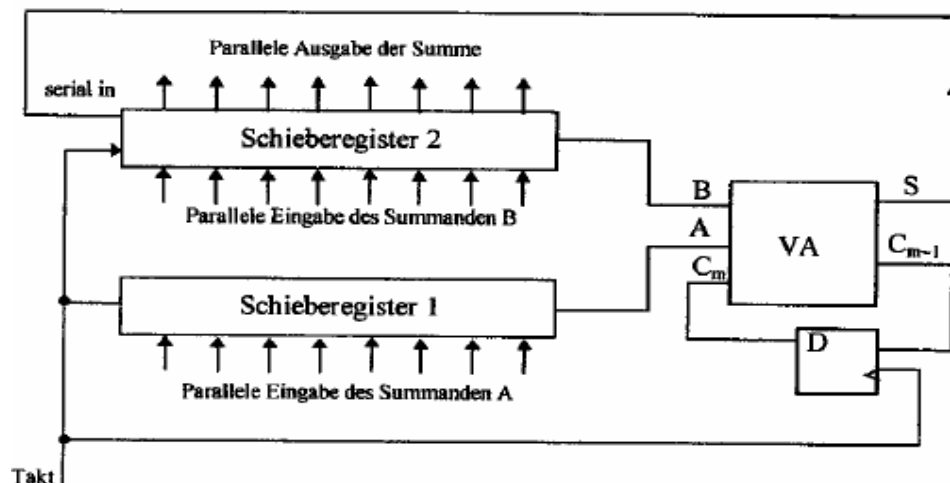


Füllen Sie die Makros aus und testen Sie die Schaltung.

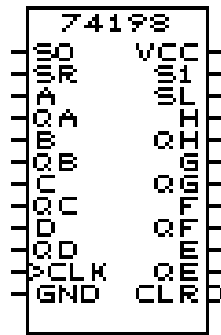
Versuch 630 Operandenregister

Im Addierwerk des Typs Ripple-Carry-Adder arbeiten die Volladdierer parallel, d.h. gleichzeitig. Die von den Volladdierern berechneten Summen stehen aber nicht zur gleichen Zeit zur Verfügung, weil jeder der Volladdierer des Ripple-Carry-Adders einen Übertrag von der nächstniedrigeren Stelle erhält. Die Summenwerte an den Ausgängen des Addierwerks sind erst dann gültig, wenn der Volladdierer der Stelle mit dem höchsten Gewicht (2^{n-1}) den Übertrag C_{n-1} erhalten hat. M.a.W.: die Überträge entstehen nacheinander. Zwei Gatterverzögerungszeiten nach Eingabe der Summanden A und B entstehen zuerst der Übertrag C_1 , dann nach weiteren zwei Gatterverzögerungszeiten der Übertrag C_2 usw. Erst wenn der Übertrag C_{n-1} nach $2n$ Gatterverzögerungszeiten entstanden ist, steht nach einer weiteren Gatterverzögerungszeit die Ziffer der höchsten Stelle der Summe zur Verfügung. In diesem Sinn arbeitet der Ripple-Carry-Adder auch seriell.

In einigen der folgenden Experimente werden wir mit der Carry-Look-Ahead-Schaltung eine Möglichkeit kennen lernen, diesen seriellen Prozess zu parallelisieren. In diesem Experiment soll ein vollkommen seriell arbeitendes Addierwerk untersucht werden, dessen Arbeitsprinzip die nächste Abbildung darstellt. Den beiden Schieberegister 1 und 2 werden über parallele Eingaben die Summanden A und B eingegeben. Es gibt nur einen Volladdierer. Wenn die beiden Summanden parallel eingegeben worden sind, werden sie Stelle für Stelle nach rechts in den Volladdierer geschoben. Mit jedem Schiebevorgang gelangt der unmittelbar vorher entstandene Summenwert S über den seriellen Eingang in das Schieberegister 2. Wenn die beiden Summanden vollkommen herausgeschoben sind, steht das Ergebnis $A + B$ im Schieberegister 2 und ist damit an dessen Parallelausgängen verfügbar.



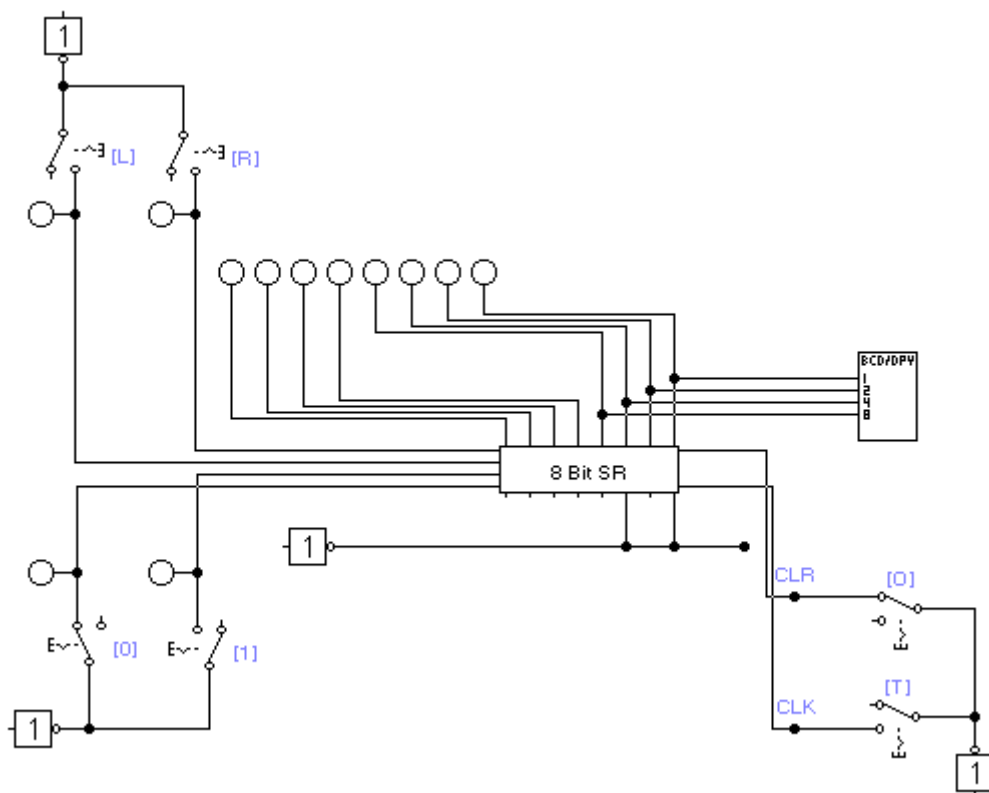
Sie könnten jetzt das Universalschieberegister, das Sie in einem der vorhergehenden Experimente gebaut haben, verwenden. Statt dessen wollen wir hier das standardisierte Industriebauelement der Typenreihe 74 benutzen. Drücken Sie den letzten Button der EWB-Bauelementebibliothek. Ziehen Sie das IC-Symbol mit der Aufschrift 741xx in das Arbeitsfeld. Wenn Sie auf dieses Symbol doppelklicken erreichen Sie den Käfer 74198 (8-Bit-Schieberegister):



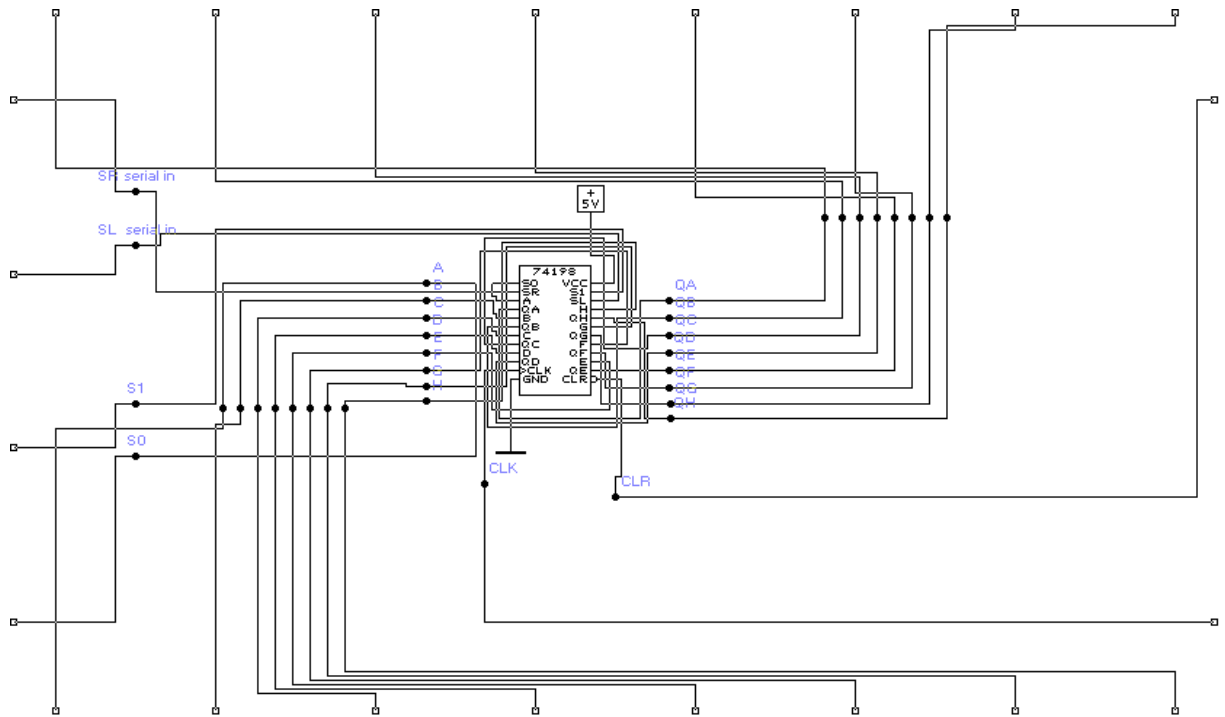
Die folgende Tabelle ist eine vereinfachte Funktionstabelle dieses Schaltkreises:

S0	S1	CLK	CLR	SL	SR	Erläuterung
0	0	↑	1	X	X	Speichern
0	1	↑	1			Linksschieben, serielle Eingabe über Eingang SL
1	0	↑	1			Rechtsschieben, serielle Eingabe über Eingang SR
1	1	↑	1	X	X	Parallele Eingabe
X	X	X	0	X	X	Löschen (bei CLR=0!)

Das Bauelement 74198 im EWB kann man nicht drehen. Deshalb befindet sich in der Datei v630 das Makro 8 Bit SR:



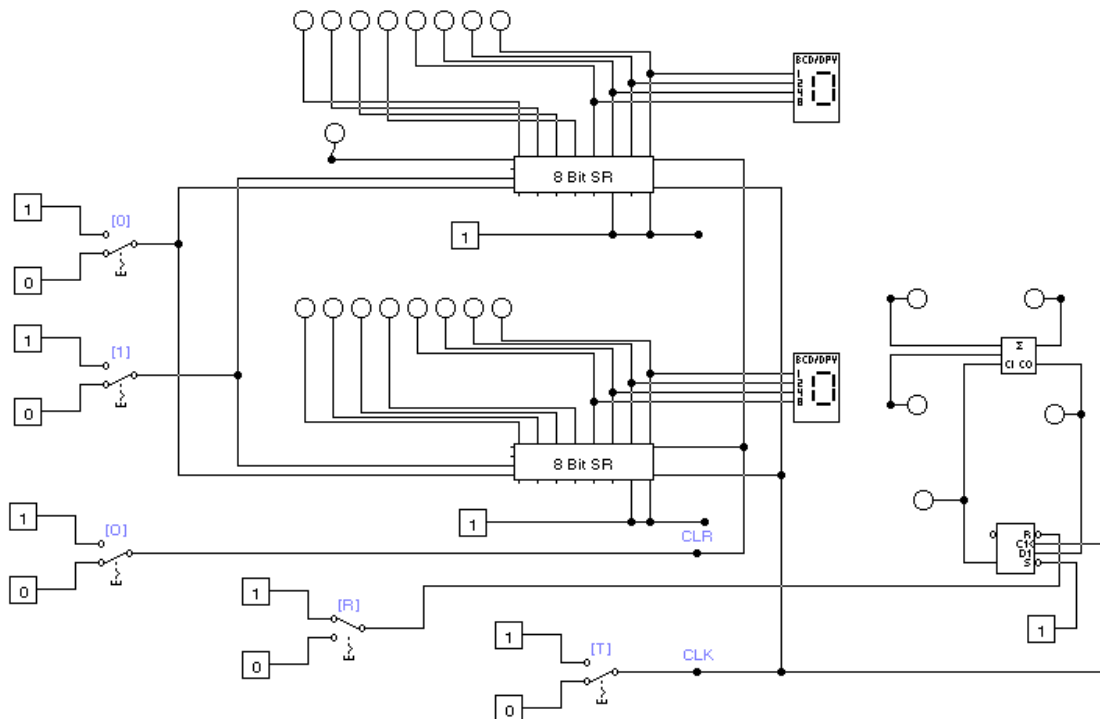
Das Innenleben sieht folgendermaßen aus:



Ergründen Sie die Arbeitsweise des Schieberegisters 74198.

Versuch 640 Serielles Addierwerk

In der Datei v640 finden Sie ein mit zwei Schieberegistern 74198 aufgebautes serielles Addierwerk:



In dieser Schaltung fehlen noch **drei** Leitungen. Machen Sie die Schaltung durch Einbau der fehlenden Leitungen funktionsfähig. Sie können die Schaltung mit Summanden der Form 0000uvwx testen, d.h. mit Summanden, deren ersten vier Stellen auf 0 gesetzt sind.

Bedienung:

Löschen Sie die Register mit einem Impuls durch Schalter O.

Legen Sie die Operanden an die unteren Paralleleingänge der Schieberegister an, für eine 1 mit der darunterliegenden horizontalen Leitung verbinden, für eine 0 offen lassen. In obiger Schaltung ist beispielsweise der Operand 00000011 angelegt.

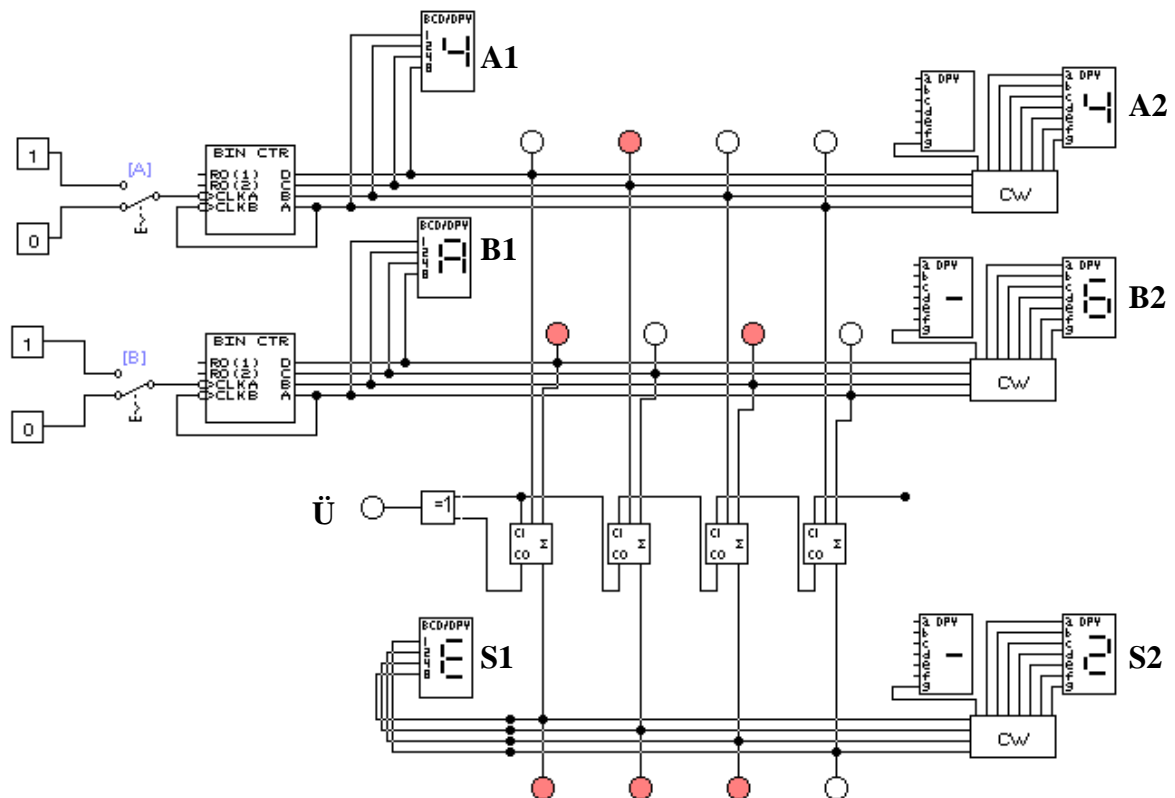
Schalten Sie den Paralleleingabe-Modus für die Register ein und takten Sie einmal.

Stellen Sie auf Rechtsschieben um.

Löschen Sie das Übertragsflipflop durch einen Impuls mit Schalter R.

Takten Sie für die Addition 8 mal.

Versuch 645 Addierwerk für ganze Zahlen

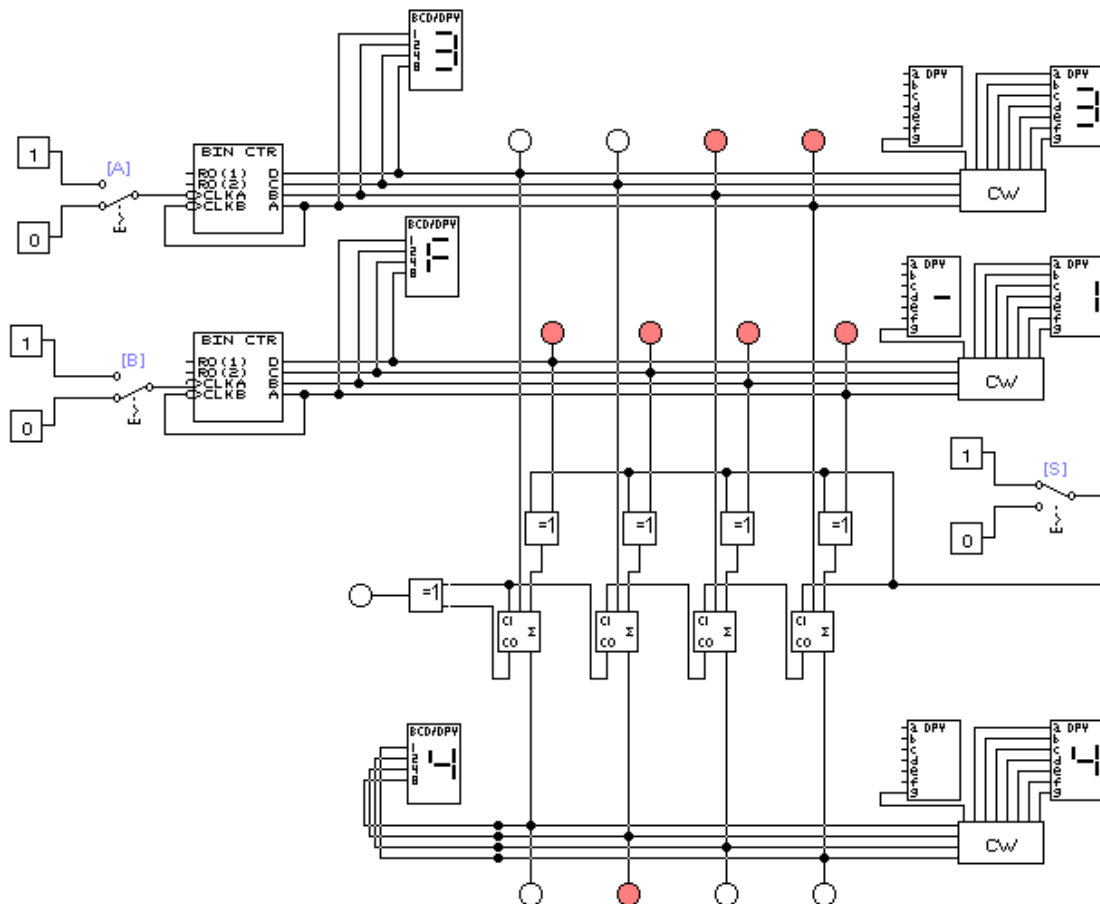


Die obige Schaltung befindet sich in der Datei v645 und enthält ein Addierwerk des Typs Ripple-Carry-Adder. Wir verwenden hier aber nicht Ihren selbstgebauten Volladdierer, sondern vier Exemplare der Volladdiererblackbox aus der EWB-Bauelementebibliothek. Die Operanden (Summanden) A und B werden mit den Zählern BIN CTR erzeugt. Neben der bereits bekannten Sieben-Segment-Anzeige die die Hexadezimalziffern zeigen kann, benutzen wir die Universal-Sieben-Segment-Anzeige mit den sieben balkenförmigen Leuchtdioden a, b, c, d, e, f, g, die über die Eingänge gleichen Namens ansteuerbar sind. Wenn am Anschluss mit dem Namen x der Binärzustand 1 herrscht, dann leuchtet in EWB die balkenförmige Leuchtdiode mit dem Namen x. Im Makro CW befindet sich ein Codewandler.

- Experimentieren Sie mit der Schaltung. Um die Schaltung zu verstehen müssen Sie die Zweikomplementdarstellung verstehen.
- Was wird in den Anzeigen A1, B1 und S1 hexadezimal angezeigt?
- Was wird in den Anzeigen A2, B2 und S2 dezimal angezeigt?
- Was zeigt die Lampe \ddot{U} an?
- Beim experimentieren sollten Sie erkennen, dass der Ripple-Carry-Adder ganze Zahlen addiert, wenn man die eingegebenen Summanden und die ausgegebenen Summe als Zweikomplementdarstellung ganzer Zahlen interpretiert. Analysieren Sie die Wirkungsweise der Schaltung.

Versuch 650 Subtrahierwerk

Die nächste Schaltung, die Sie in der Datei v650 finden, ist fast eine Kopie der Schaltung aus dem vorherigen Versuch. Wenn Sie mit dem Schalter S den Binärzustand 0 eingeben, hat die Schaltung das gleiche Verhalten wie die vorherige Schaltung.



- Durch den Schalter S legen Sie fest, ob die Schaltung Addieren (S=0) oder Subtrahieren (S=1) soll. Schalten Sie S=1.
- Beim Experimentieren sollten Sie nun erkennen, dass die Schaltung die Differenz Eingabe A – Eingabe B in Zweikomplementdarstellung erzeugt.
- Erklären Sie die Wirkungsweise der Schaltung. Hinweise:
Jede Subtraktion $A - B$ lässt sich als Addition durchführen: $A - B = A + (-B)$. Das Zweierkomplement von $(-B)$ ist $(\neg B) + 1$. Darin ist $(\neg B)$ die stellenweise Invertierung von B. Erklären Sie, wie in der Schaltung die stellenweise Invertierung und das Inkrement realisiert werden.

Theorie 657 Carry Look Ahead Adder

Je größer die Stellenzahl des Ripple-Carry-Adders, desto länger dauert es, bis das vollständige Ergebnis verfügbar ist. Man könnte versuchen, die Addierwerke in Normalform zu entwickeln. Dann müssten die Signale über maximal drei Gatter laufen. Das würde aber z.B. für ein 8-Bit-Addierwerk eine Funktionstabelle mit $2^{17} = 131072$ Zeilen erfordern. Lange Signallaufzeiten entstehen im Ripple-Carry-Adder durch die Carry-Signale. Die Idee des Carry-Look-Ahead-Adders ist es, die Carry-Signale nicht mehr von Adder-Modul zu Adder-Modul weiter zu reichen, sondern in einer zusätzlichen kombinatorischen Schaltung direkt aus den Eingangsgrößen A_n und B_n zu erzeugen, die Signale dabei über möglichst weniger Gatter laufen zu lassen und alle Carry-Signale nach der gleichen Verzögerungszeit zu gewinnen.

Als Beispiel wählen wir ein vierstelliges Addierwerk. Die Carrys erhalten hierbei den Index der Stelle, in die sie einfließen.

Ein Addierwerk berechnet folgende Summen und Carry-Werte:

$$S_0 = A_0 \text{ XOR } B_0 \text{ XOR } C_0$$

$$S_1 = A_1 \text{ XOR } B_1 \text{ XOR } C_1$$

$$S_2 = A_2 \text{ XOR } B_2 \text{ XOR } C_2$$

$$S_3 = A_3 \text{ XOR } B_3 \text{ XOR } C_3$$

$$C_1 = (A_0 \wedge B_0) \vee (A_0 \vee B_0) \wedge C_0$$

$$C_2 = (A_1 \wedge B_1) \vee (A_1 \vee B_1) \wedge C_1$$

$$C_3 = (A_2 \wedge B_2) \vee (A_2 \vee B_2) \wedge C_2$$

$$C_4 = (A_3 \wedge B_3) \vee (A_3 \vee B_3) \wedge C_3$$

Wir führen zwei Hilfsvariablen g_n und p_n ein:

$$g_n = A_n \wedge B_n \quad p_n = A_n \vee B_n$$

- g_n heisst Carry generate weil ein Übertrag C_n gebildet wird, denn sowohl A_{n-1} als auch B_{n-1} den Binärzustand 1 haben. In diesem Fall wird also allein vom A_{n-1} – und B_{n-1} – Wert ein Übertrag generiert, unabhängig vom C_{n-1} – Wert.
- p_n heisst Carry propagate weil der Übertrag C_{n-1} weitergeleitet wird, wenn $p_n = 1$ und $g_n = 0$ ist.

... und setzen sie ein: Zur Erinnerung: Ohne Klammerung bindet \wedge stärker als \vee .

$$C_1 = g_0 \vee p_0 \wedge C_0$$

$$C_2 = g_1 \vee p_1 \wedge C_1$$

$$C_3 = g_2 \vee p_2 \wedge C_2$$

$$C_4 = g_3 \vee p_3 \wedge C_3$$

Nach Ersetzen von C_1 , C_2 und C_3 auf den rechten Seiten:

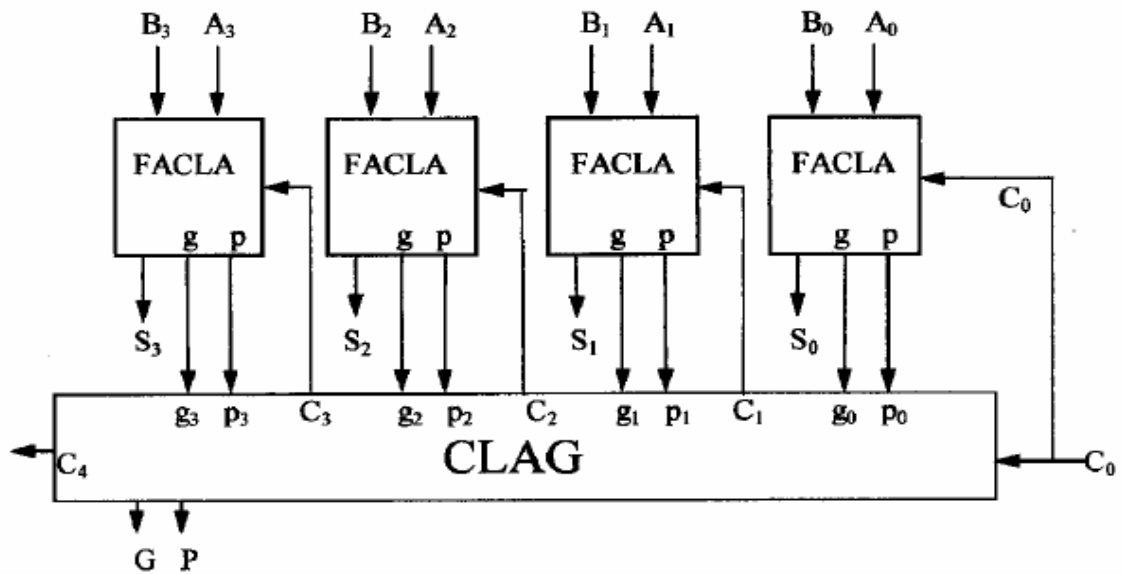
$$C_1 = g_0 \vee p_0 \wedge C_0$$

$$C_2 = g_1 \vee p_1 \wedge g_0 \vee p_1 \wedge p_0 \wedge C_0$$

$$C_3 = g_2 \vee p_2 \wedge g_1 \vee p_2 \wedge p_1 \wedge g_0 \vee p_2 \wedge p_1 \wedge p_0 \wedge C_0$$

$$C_4 = g_3 \vee p_3 \wedge g_2 \vee p_3 \wedge p_2 \wedge g_1 \vee p_3 \wedge p_2 \wedge p_1 \wedge g_0 \vee p_3 \wedge p_2 \wedge p_1 \wedge p_0 \wedge C_0$$

Wenn man nun die Volladdierschaltung so umbaut, dass sie neben der Summe S_n auch die Hilfsvariablen g_n und p_n liefern, kann man einen n-stelligen Carry-Look-Ahead-Adder bauen:



Die Blackboxes FACLA erhalten die umgebaute Volladdiererschaltung. CLAG ist der Carry-Look-Ahead-Generator. Er erzeugt aus den g- und p-Hilfsvariablen die Überträge C_n . Man beachte: Die in dieser Schaltung benutzten Volladdierer erzeugen keine Überträge. Über die Ein/Ausgänge C_0 , C_4 , G und P können mit mehreren CLAGs mehrstufige Carry-Look-Ahead-Generatoren erzeugt werden:

$$C_4 = g_3 \vee p_3 \wedge g_2 \vee p_3 \wedge p_2 \wedge g_1 \vee p_3 \wedge p_2 \wedge p_1 \wedge g_0 \vee p_3 \wedge p_2 \wedge p_1 \wedge p_0 \wedge C_0 \text{ mit}$$

$$G = g_3 \vee p_3 \wedge g_2 \vee p_3 \wedge p_2 \wedge g_1 \vee p_3 \wedge p_2 \wedge p_1 \wedge g_0 \text{ und}$$

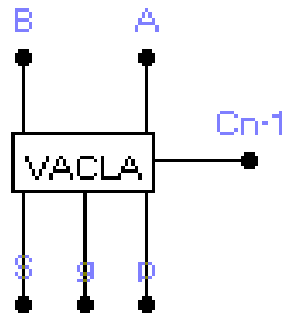
$$P = p_3 \wedge p_2 \wedge p_1 \wedge p_0.$$

$$\text{Also } C_4 = G \vee P \wedge C_0$$

Versuch 658 Volladdierer für Carry-Look-Ahead Adder

Bauen Sie in einer leeren Datei v658 die oben beschriebene Addierschaltung FACLA durch Modifikation einer Volladdierschaltung. Die Schaltung soll eine möglichst geringe Tiefe haben. Sie dürfen beliebige Gattertypen mit auch mehr als zwei Eingängen benutzen.

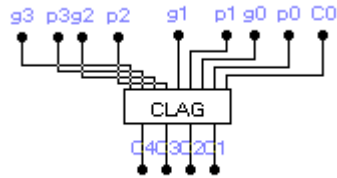
Verpacken Sie Ihre Schaltung in ein Makro mit der folgenden Anschlußbelegung:



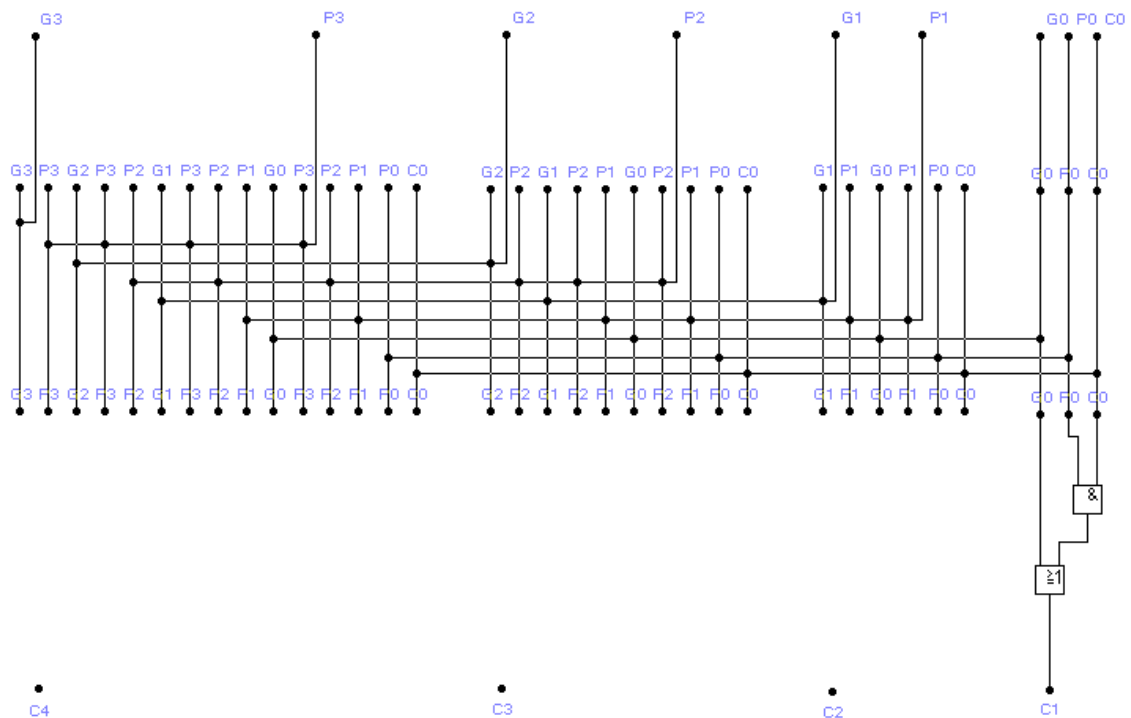
Die Bezeichnungen FACLA und VACLA sind synonym.

Versuch 660 Carry-Look-Ahead-Generator

Bauen Sie in der Datei v660 einen vierstelligen Carry-Look-Ahead-Generator in ein Makro mit der folgenden Anschlußbelegung:



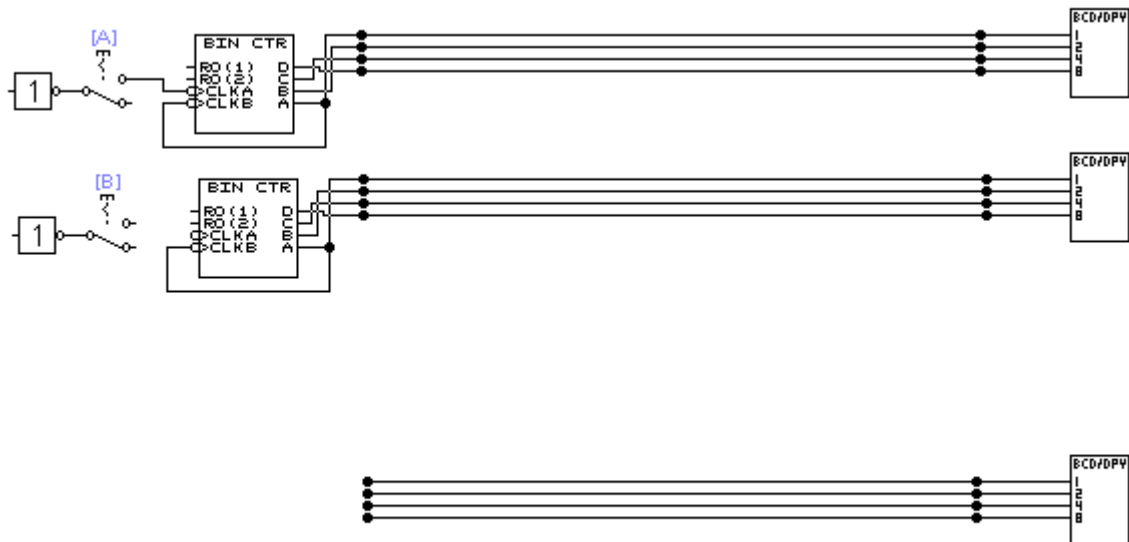
In der gleichen Datei finden Sie das folgende Verdrahtungsschema vorgegeben:



Ergänzen Sie dieses Verdrahtungsschema durch ein Schaltnetz zum Carry-Look-Ahead-Generator. Auch diese Schaltung soll eine möglichst geringe Tiefe haben.

Versuch 665 Carry-Look-Ahead Addierwerk

Bauen Sie in der Datei v665 mit dem Volladdierer und dem Carry-Look-Ahead-Generator aus den vorherigen Versuchen ein vierstelliges Carry-Look-Ahead-Addierwerk in folgender Meßschaltung:



Testen Sie die funktionsfähige Schaltung.

Die Kaskadiereigenschaft müssen Sie nicht implementieren.

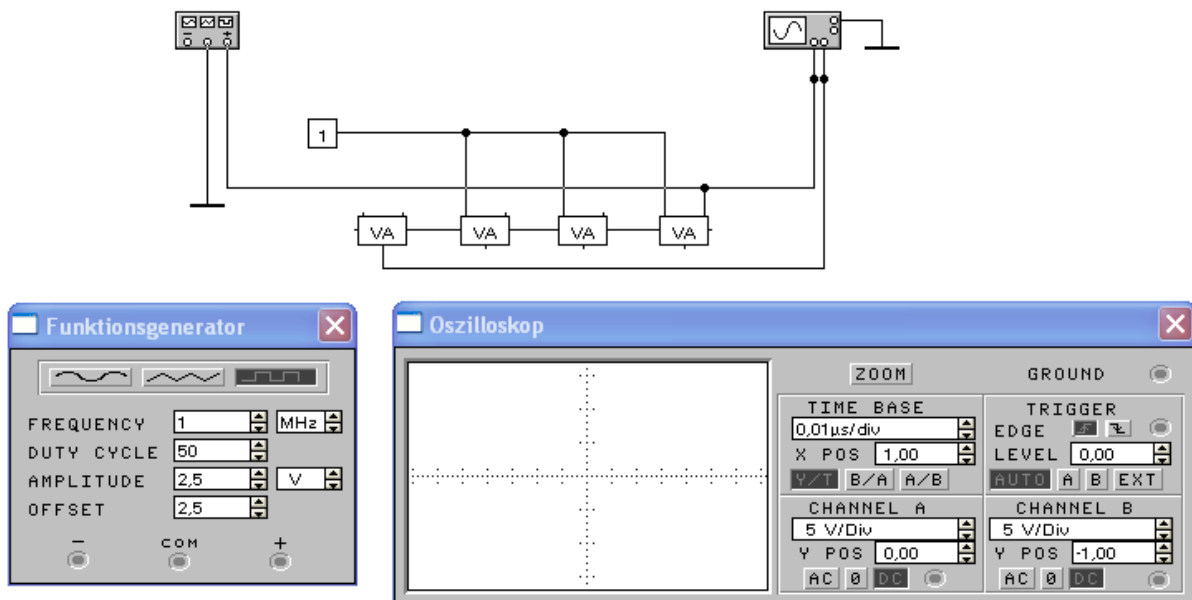
Versuch 670 Verzögerungszeit im Ripple-Carry-Adder

Im Ripple-Carry-Adder wird der Übertrag von Volladdierer zu Volladdierer von links nach rechts durchgereicht. Die höchste Stelle des Summanden ist erst dann gültig, wenn dieser Durchreichvorgang beendet ist. Wir wollen nun Messen, wieviel Zeit dieser Vorgang in einem mit EWB-Gattern aufgebauten vierstelligen Ripple-Carry-Adder dauert.

Um eine Schaltung mit möglichst geringer Tiefe zu erhalten, wurde die Aufbauvariante:

$$C_{n+1} = (A_n \wedge B_n) \vee (A_n \wedge C_n) \vee (B_n \wedge C_n) \quad \text{gewählt.}$$

Messen Sie in der Datei v670, wieviel Zeit der Ripple-Carry-Adder zur Erzeugung des Summenbits an der höchstwertigen Stelle benötigt (Mitte Flanke bis Mitte Flanke). Die Messgenauigkeit soll eine Gatterlaufzeit = 10 ns betragen.



Bei der Messung wird folgende Additionsaufgabe gelöst:

$$\begin{array}{r} 0111 \\ + 0001 \\ \hline 1000 \end{array}$$

Messergebnis: ___ ns

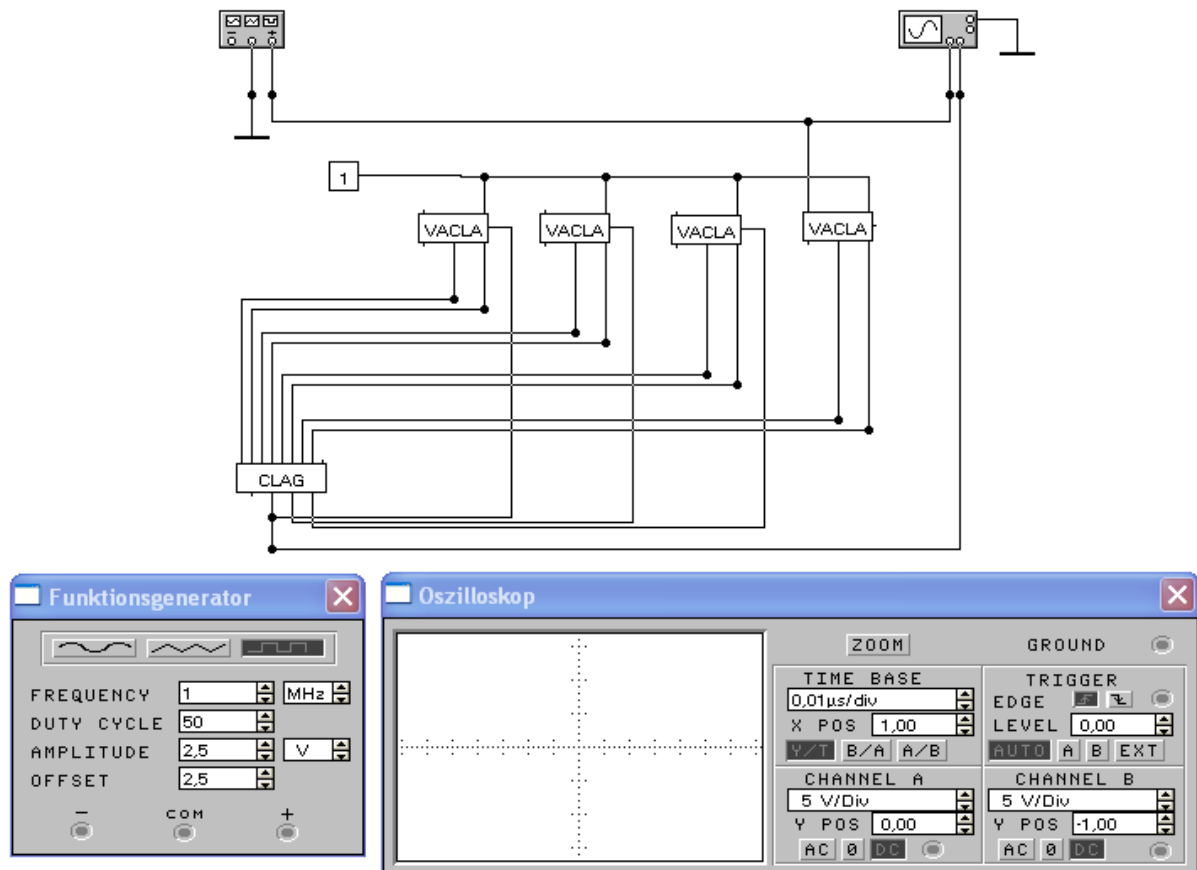
Hinweis: Falls die Kurven nur unvollständig gezeichnet werden, haben Sie vergessen, wieder auf Transientenanalyse zurückzustellen (F3 drücken und auswählen).

Wieviel länger würde ein fünfstelliger Ripple-Carry Adder brauchen?

Versuch 675 Verzögerungszeit im Carry-Look-Ahead-Adder

Fügen Sie Ihre Makros für VACLA und CLAG ein. Falls noch nicht geschehen, benutzen Sie zur Summenbildung ein dreifach-XOR.

Messen Sie in der Datei v675, wieviel Zeit der Carry-Look-Ahead-Adder benötigt, um den Übertrag C_3 zu erzeugen (Mitte Flanke bis Mitte Flanke). Die Messgenauigkeit soll eine Gatterlaufzeit = 10 ns betragen.



Bei der Messung wird folgende Additionsaufgabe gelöst:

```
  0111
+ 0001
-----
 1000
```

Messergebnis: ____ ns

Wieviel länger würde ein fünfstelliger CLA Adder brauchen?

Theorie 678 Multiplikation

Im Folgenden werden Maschinen, die nach dem bürgerlichen Multiplikationsalgorithmus arbeiten, untersucht. Beispiel einer Multiplikation nach diesem Verfahren:

$$1101 \cdot 1011 = X \cdot Y = X \cdot y_3 y_2 y_1 y_0$$

$$1101 = 2^0 \cdot X \cdot y_0$$

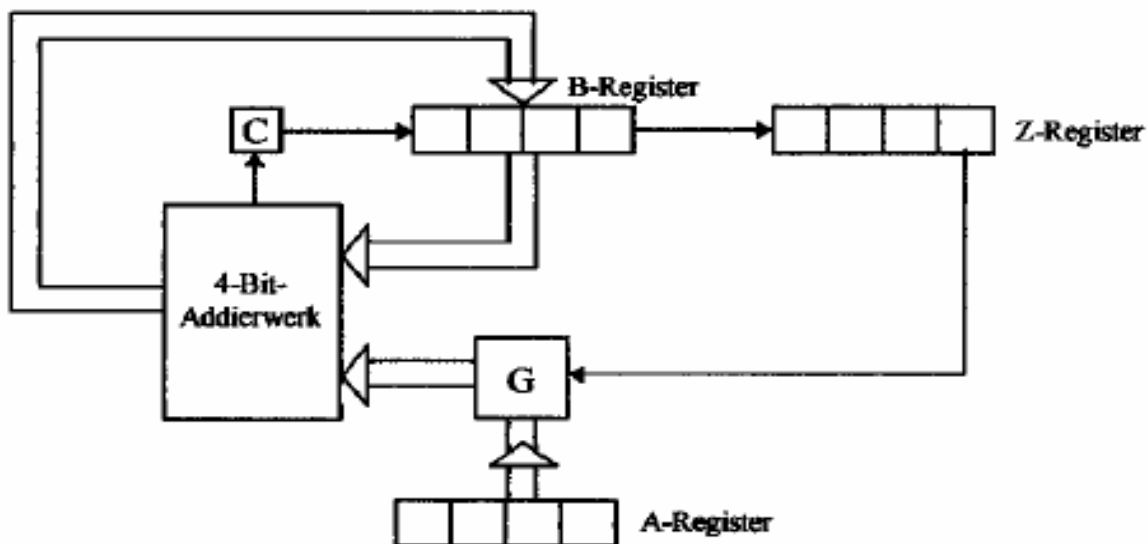
$$1101 = 2^1 \cdot X \cdot y_1$$

$$0000 = 2^2 \cdot X \cdot y_2$$

$$1101 = 2^3 \cdot X \cdot y_3$$

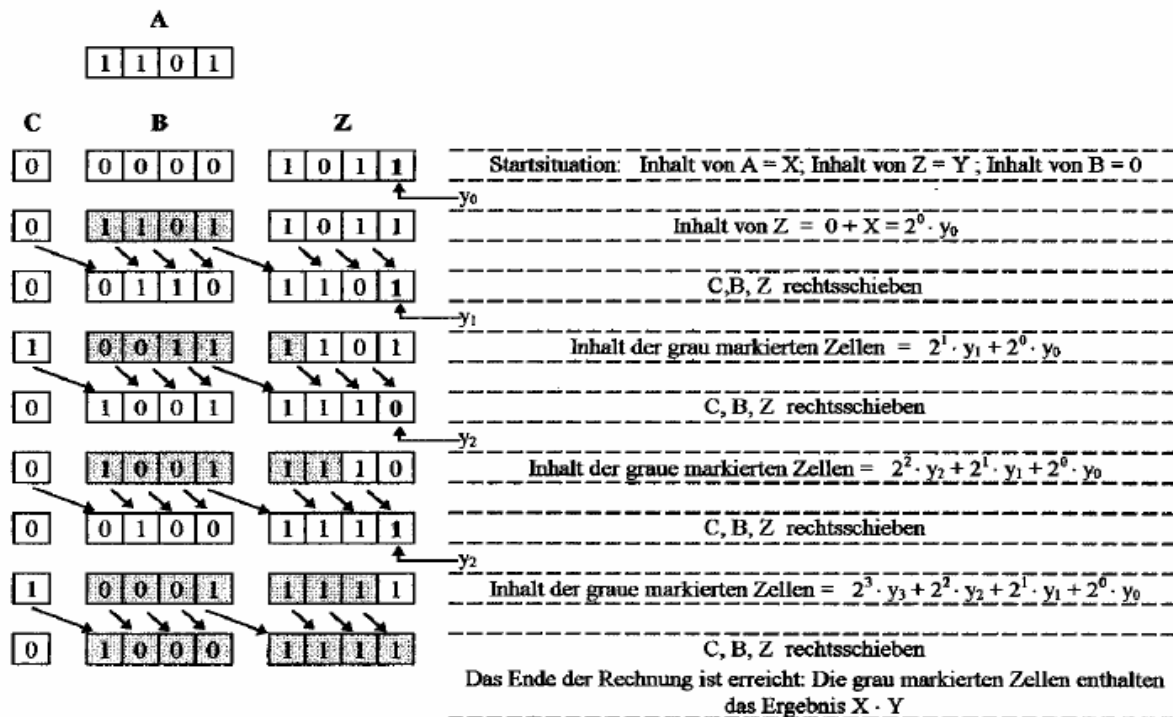
$$10001111 = X \cdot Y = \sum_{i=0}^3 2^i \cdot X \cdot y_i$$

Ein seriell arbeitendes vierstelliges Multiplizierwerk lässt sich nach folgendem Schema aufbauen:



Arbeitsweise:

Die Register B und Z sind hintereinandergeschaltete Schieberegister. Sie bilden zusammen mit dem Flip-Flop C (Carry) ein 9-Bit-Schieberegister, in dem nach rechts geschoben werden kann. Das Schieberegister A ist parallel ladbar und parallel auslesbar. Das Schieberegister Z ist parallel auslesbar. A ist ein einfaches Register (d.h. kein Schieberegister). Die Parallelausgänge von Z sind in der obigen Abbildung nicht dargestellt. Die Blackbox G enthält vier UND-Gatter, mit der die Eingabe in das Addierwerk gesteuert wird. Wenn im letzten Flip-Flop des Schieberegisters Z der Binärzustand 1 steht, wird der Inhalt von Register A in das Addierwerk eingegeben und sonst, d.h. wenn das letzte Flip-Flop des Registers Z den Binärzustand 0 enthält, gibt die Blackbox G eine Null in das Addierwerk ein. Zu Beginn der Rechnung steht der erste Faktor X in Register A und der zweite Faktor Y im Register Z. Die Rechenschritte bei der Berechnung des Produktes $1101 \cdot 1011$ können Sie mit Hilfe des folgenden Protokolls nachvollziehen:



In einem der vorhergehenden Experimente wurde ein seriell Addierwerk simuliert, dem ein ähnliches Schaltungsschema zu Grunde lag, wie diesem seriell arbeitenden Multiplizierwerk. Wir simulieren den seriellen Multiplizierer deshalb nicht und wenden uns im Folgenden einer mehr parallel arbeitenden Multiplizierschaltung zu, die als zweidimensionales Array aufgebaut ist. Bevor wir diese Schaltung entwickeln, sehen wir uns noch einmal genau den Algorithmus an, nach dem das obige serielle Multiplizierwerk arbeitet. Er unterscheidet sich vom ursprünglichen bürgerlichen Multiplizieralgorithmus, in dem zuerst die Produkte $2^i \cdot X \cdot y_i$ gebildet werden, im letzten Schritt die Summe dieser Produkte:

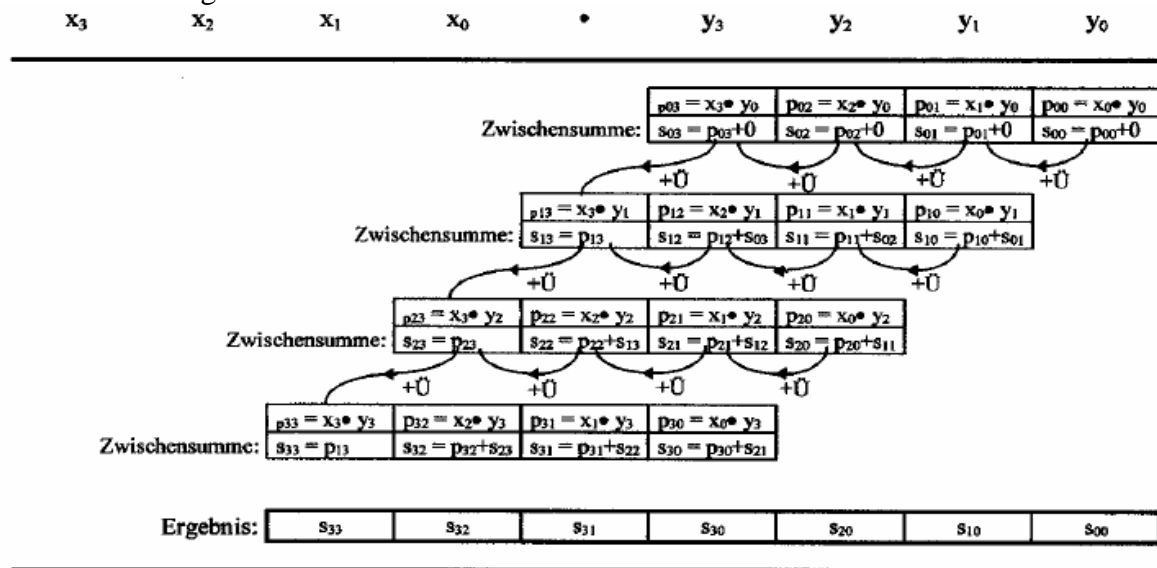
Schritte im ursprünglichen Multiplikationsalgorithmus:

1. Schritt: $P_0 = 2^0 \cdot X \cdot y_0 = 1101$
2. Schritt: $P_1 = 2^1 \cdot X \cdot y_1 = 11010$
3. Schritt: $P_2 = 2^2 \cdot X \cdot y_2 = 000000$
4. Schritt: $P_3 = 2^3 \cdot X \cdot y_3 = 1101000$
5. Schritt: $P_4 = \sum_{i=0}^3 2^i \cdot X \cdot y_i = 10001111$

Im seriellen Addierwerk wurden Zwischensummen gebildet:

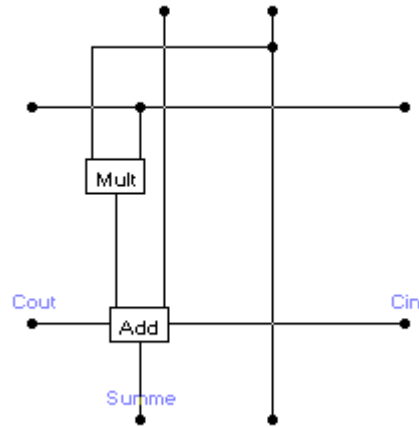
1. Schritt: $P_0 = 0000$
2. Schritt: $2^0 \cdot X \cdot y_0 = 1101$
3. Schritt: $P_1 = P_0 + 2^0 \cdot X \cdot y_0 = 1101$ (Zwischensumme)
4. Schritt: Rechtsschieben von $P_1 = 01101$
 $2^1 \cdot X \cdot y_1 = 11010$
5. Schritt: $P_2 = P_1 + 2^1 \cdot X \cdot y_1 = 100111$ (Zwischensumme)
6. Schritt: Rechtsschieben von $P_2 = 100111$
 $2^2 \cdot X \cdot y_2 = 000000$
7. Schritt: $P_3 = P_2 + 2^2 \cdot X \cdot y_2 = 100111$ (Zwischensumme)
8. Schritt: Rechtsschieben von $P_3 = 0100111$
 $2^3 \cdot X \cdot y_3 = 1101000$
9. Schritt: $P_4 = P_3 + 2^3 \cdot X \cdot y_3 = 10001111$ (Ergebnis)


Auch der als zweidimensionales Array aufgebaute Multiplexer bildet Zwischensummen. Er arbeitet nach folgendem Schema:

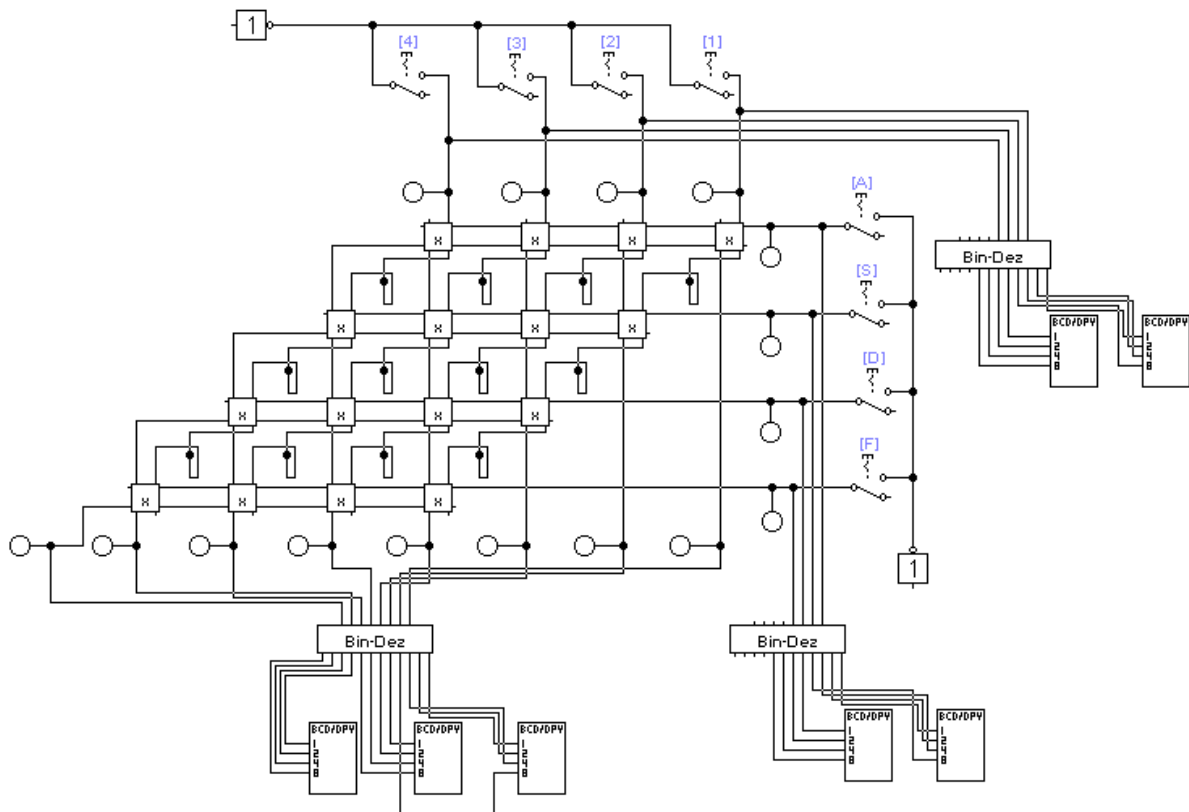


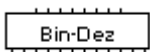
Versuch 680 Entwicklung eines Prozessors für ein Multiplikationswerk

Wir wollen in diesem Versuch ein vierstelliges Multiplikationswerk für das duale Ziffernsystem $\{0,1\}$ entwickeln. Dafür brauchen wir einen Prozessor mit vier Ein- und vier Ausgängen:



- Die Blackbox muss in diesem Fall das Einmaleins beherrschen:
 $0 \cdot 0 = 0$
 $0 \cdot 1 = 0$
 $1 \cdot 0 = 0$
 $1 \cdot 1 = 1$
Welches Gatter berechnet diese Funktion?
- Entwickeln Sie diesen Prozessor als Digitalschaltung und bauen Sie ihn dann in der Datei v680 in das Makro  ein. Wenn Sie alles richtig gemacht haben, müsste die Schaltung multiplizieren.



Das Makro  enthält einen Codewandler, der Dualzahlen in BCD-Zahlen umformt, die dann von den BCD-kodierten Sieben-Segment-Anzeigen als Dezimalzahlen angezeigt werden. Sie können also die Faktoren $X = x_3 \ x_2 \ x_1 \ x_0$ und $Y = y_3 \ y_2 \ y_1 \ y_0$ mit den Tastenschaltern als Dualzahlen eingeben und als Dezimalzahlen an die BCD-kodierten Sieben-Segment-Anzeigen ablesen. Auch das berechnete Produkt ist als Dezimalzahl ablesbar.