Kapitel 6 - Addierwerke

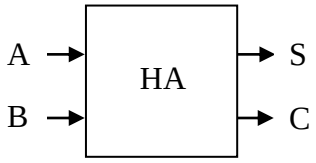
Versuch 600 Halbaddierer und Volladdierer

Der bürgerliche Algorithmus des schriftlichen Addierens zerlegt die binäre Addition in die folgenden elementaren Additionen. Es ergibt sich eine Summe S und ein Carry C (Übertrag) mit der zugehörigen Funktiontabelle:

A + B = S, C

0 + 0 = 0, C = 0
0 + 1 = 1, C = 0
1 + 0 = 1, C = 0
1 + 1 = 0, C = 1

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



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:

(1) $S = (A \wedge \overline{B}) \vee (\overline{A} \wedge B)$
(2) $C = A \wedge B$

Die Gleichung (1) entspricht der XOR-Funktion. Der Halbaddierer lässt sich also durch die beiden folgenden Gleichungen beschreiben:

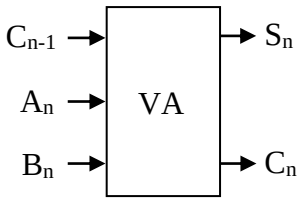
(3) S = A XOR B
(4) $C = A \wedge B$

Für die Addition zweier mehrstelliger Binärzahlen müssen drei Binärziffern addiert werden können: der Übertrag von der vorhergehenden Stelle und die beiden Summanden. Nur in der niederwertigsten Stelle (LSB) gibt es keinen Übertrag. Es ergibt sich folgende Funktiontabelle:

$A_n + B_n + C_{n-1} = S_n , C_n$

$0 + 0 + 0 = 0,\ C_n = 0$
$0 + 0 + 1 = 1,\ C_n = 0$
$0 + 1 + 0 = 1,\ C_n = 0$
$0 + 1 + 1 = 0,\ C_n = 1$
$1 + 0 + 0 = 1,\ C_n = 0$
$1 + 0 + 1 = 0,\ C_n = 1$
$1 + 1 + 0 = 0,\ C_n = 1$
$1 + 1 + 1 = 1,\ C_n = 1$

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:

(5) $S_n = (A_n \wedge \overline{B_n} \wedge \overline{C_{n-1}}) \vee (\overline{A_n} \wedge B_n \wedge \overline{C_{n-1}}) \vee (\overline{A_n} \wedge \overline{B_n} \wedge C_{n-1}) \vee (A_n \wedge B_n \wedge C_{n-1})$

Daraus kann man

(6) $S_n = A_n$ XOR B_n XOR C_{n-1}

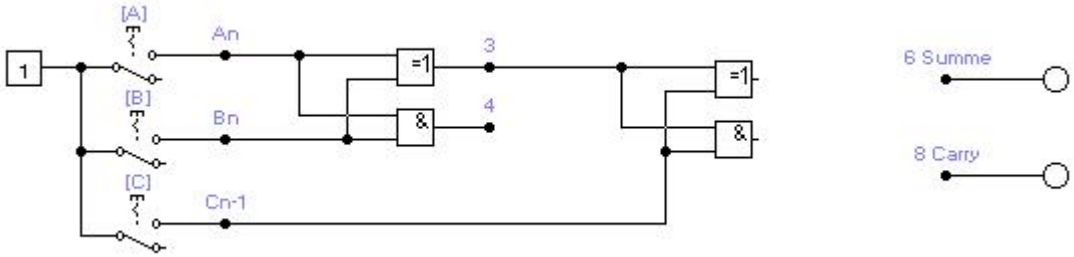
ableiten. Für C_n ergibt die Funktionstabelle die disjunktive Normalform

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

Daraus kann man die folgenden drei Varianten ableiten:

(8) $C_n = (A_n \wedge B_n) \vee ((A_n$ XOR $B_n) \wedge C_{n-1})$
(9) $C_n = (A_n \wedge B_n) \vee (A_n \wedge C_{n-1}) \vee (B_n \wedge C_{n-1})$
(10) $C_n = (A_n \wedge B_n) \vee (A_n \vee B_n) \wedge C_{n-1}$

In der Datei v600 finden Sie zwei hintereinander geschaltete Halbaddierer:

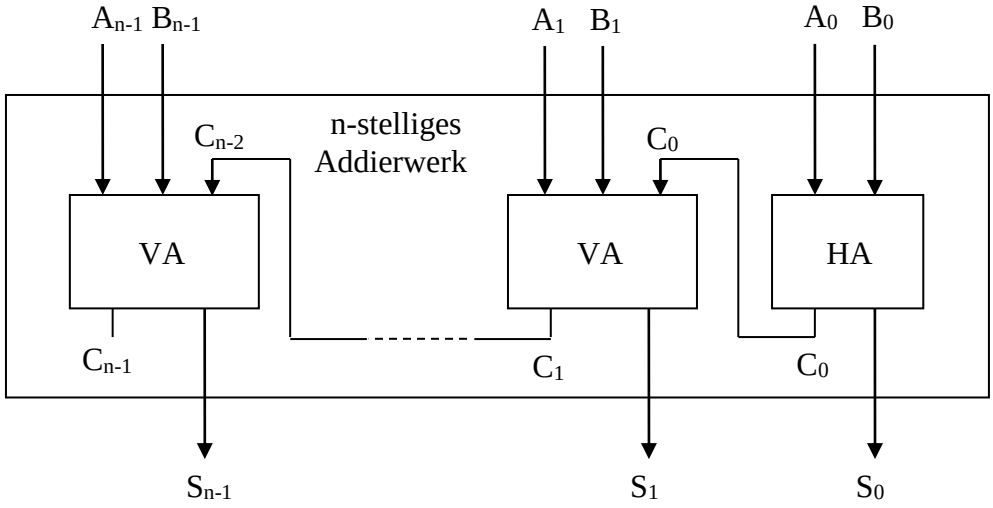


Ergänzen Sie die Schaltung mit Hilfe der beiden Gleichungen (6) und (8) so, dass ein Volladdierer entsteht.

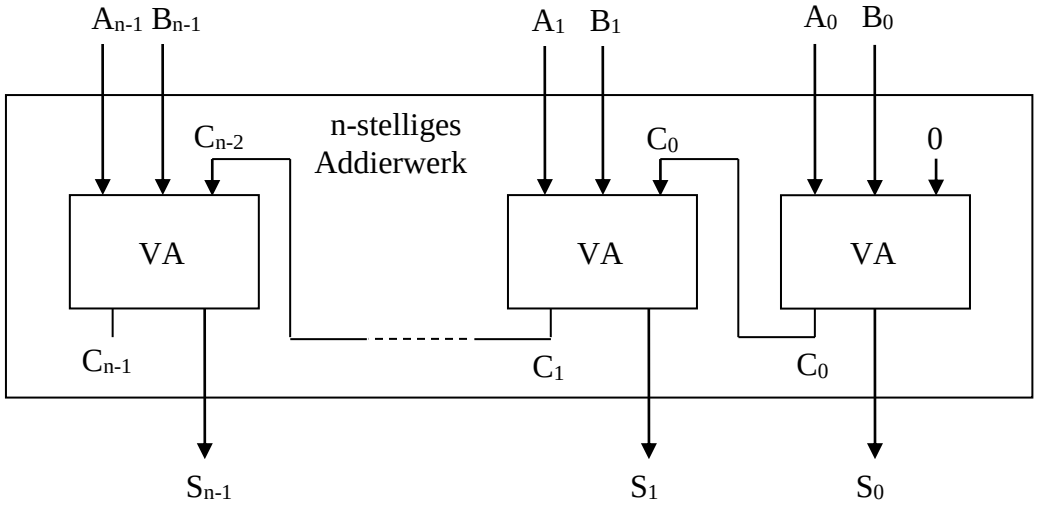
Überprüfen Sie die Schaltung auf richtige Funktionsweise.

Versuch 610 Ripple-Carry-Addierwerk für ganze Zahlen

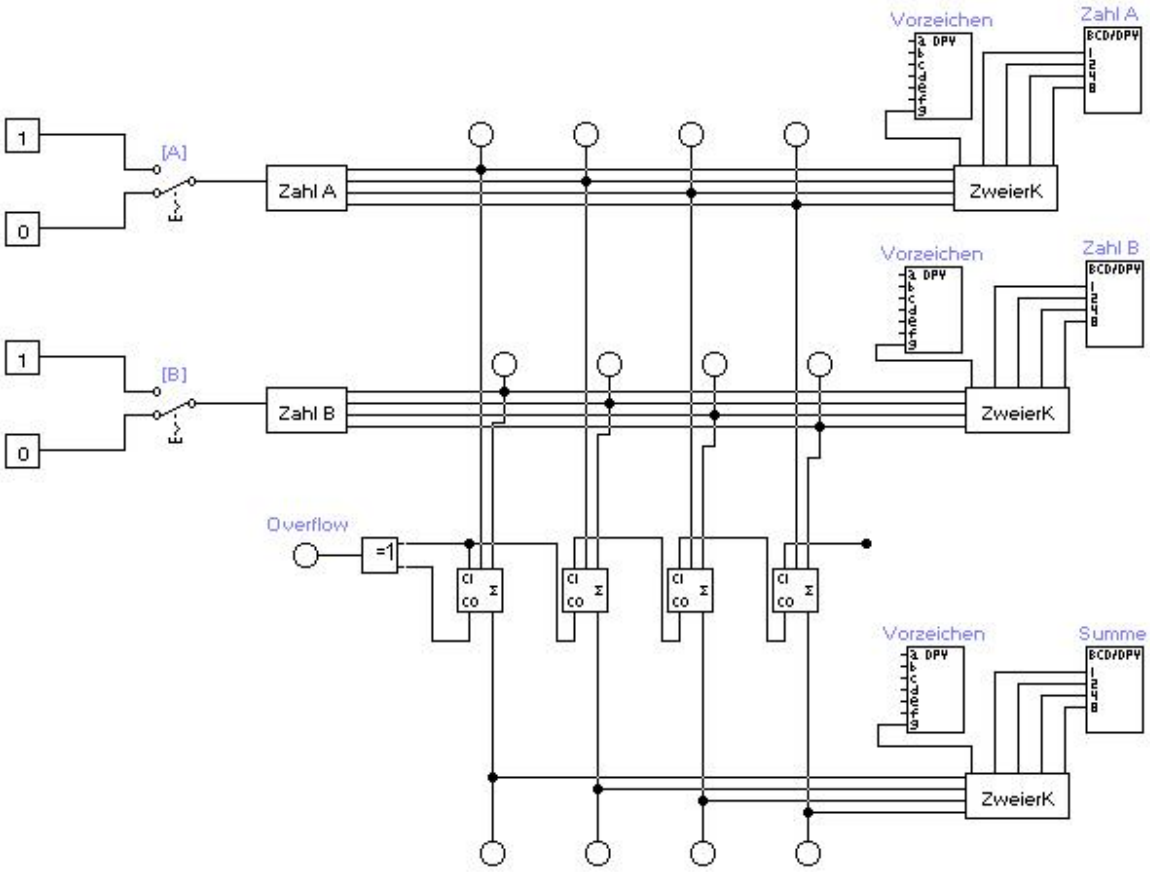
Mit n-1 Volladdierern und einem Halbaddierer kann man eine Digitalschaltung aufbauen, die zwei n-stellige Binärzahlen $A_{n-1} \ldots A_0$ und $B_{n-1} \ldots B_0$ addiert:



Ebenso lassen sich zwei n-stellige Binärzahlen addieren, wenn man nur Volladierer verwendet und den Carry-Eingang der niederwertigsten Stelle offen lässt, d.h. auf logisch 0 legt.



Im Addierwerk des Ripple-Carry-Adders 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 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. Die Überträge entstehen also nacheinander.
Erst wenn der Übertrag C_{n-1} vorliegt, steht das Ergebnis zur Verfügung. In diesem Sinn arbeitet der Ripple-Carry-Adder seriell.

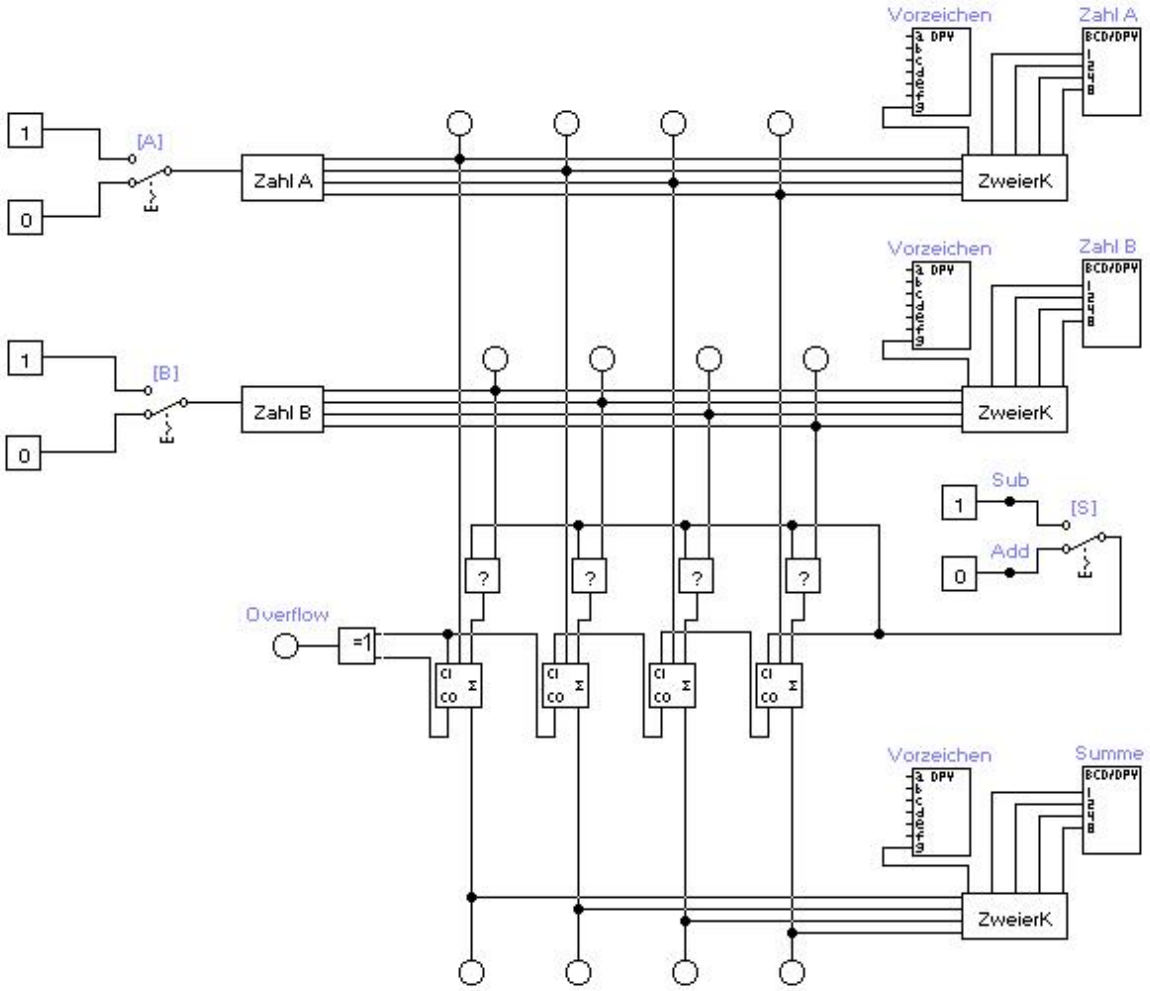


Die obige Schaltung befindet sich in der Datei v610 und enthält ein Addierwerk des Typs Ripple-Carry-Adder. Verwendet werden vorgefertigte Volladdierer aus der Bauelementebibliothek des EWB. Der Carry-In Eingang des rechten Volladdierers bleibt unbeschaltet und liegt somit auf 0.
Die Operanden (Summanden) Zahl A und Zahl B werden mit Zählern in den Makros Zahl A und Zahl B erzeugt. Im Makro ZweierK befindet sich ein Wandler, der das Vorzeichen berücksichtigt und im Anzeigeelement DPYdarstellt.

- Experimentieren Sie mit der Schaltung. Um die Schaltung zu verstehen, müssen Sie die Zweierkomplementdarstellung verstehen.
- Was wird in den Anzeigen Zahl A, Zahl B und Summe angezeigt?
- Was zeigt die Lampe Overflow an?
- Beim Experimentieren sollten Sie erkennen, dass der Ripple-Carry-Adder ganze Zahlen addiert, wenn man die eingegebenen Summanden und die ausgegebenen Summe als Zweierkomplementdarstellung ganzer Zahlen interpretiert. Analysieren Sie die Wirkungsweise der Schaltung.
- Arbeitet das Addierwerk eher seriell oder parallel bis das Endergebnis im ungünstigsten Fall vollständig anliegt?

Versuch 620 Addier- und Subtrahierwerk

Die Schaltung in v620 stellt fast eine Kopie der Schaltung aus v610 dar. Wenn Sie mit dem Schalter S den Binärzustand 0 eingeben, hat die Schaltung das selbe Verhalten wie in v610.



- Durch den Schalter S legen Sie fest, ob die Schaltung Addieren (S=0) oder Subtrahieren (S=1) soll.
- Jede Subtraktion A – B lässt sich als Addition durchführen: A – B = A + (–B). Das Zweierkomplement von (–B) ist ($\overline{B}$) + 1. Darin ist ($\overline{B}$) die bitweise Invertierung von B.
- Ergänzen Sie die Schaltung, indem Sie in das Makro „?“ durch ein geeignetes Gatter ersetzen. Eine Wertetabelle ist hilfreich, um die Funktion des Gatters zu ermitteln.
- Beim Experimentieren sollten Sie nun erkennen, dass die Schaltung die Differenz Eingabe A – Eingabe B in Zweierkomplementdarstellung erzeugt.
- Erklären Sie, wie in der Schaltung die bitweise Invertierung und das Inkrement realisiert werden.

Theorie 630 Carry-Look-Ahead Addierwerk

Ein Nachteil des Ripple-Carry-Adders ist, dass die Laufzeit bis zum vollständigen Ergebnis von der Anzahl der Stellen abhängig ist. In den folgenden Versuchen wird mit der Carry-Look-Ahead-Schaltung eine Möglichkeit aufgezeigt, diesen seriellen Prozess zu parallelisieren.
Die Idee des Carry-Look-Ahead-Adders ist es, die Carry-Signale nicht mehr von Adder-Modul zu Adder-Modul weiterzureichen, sondern in einer zusätzlichen kombinatorischen Schaltung direkt aus den Eingangsgrößen A_n und B_n zu erzeugen. Dabei sollen die Signale parallel über möglichst wenige Gatter laufen und alle Carry-Signale nach der selben Verzögerungszeit gewonnen werden.

Als Beispiel wählen wir ein vierstelliges Addierwerk, das kaskadierbar ist. Dadurch kann ein Carry C_{-1} von einem Addierwerk übernommen werden und es kann ein Carry C_3 weitergeleitet werden.

Das Addierwerk berechnet folgende Summen und Carry-Werte:

$S_0 = A_0 \text{ XOR } B_0 \text{ XOR } C_{-1}$ $\quad C_0 = (A_0 \wedge B_0) \vee (A_0 \vee B_0) \wedge C_{-1}$
$S_1 = A_1 \text{ XOR } B_1 \text{ XOR } C_0$ $\quad C_1 = (A_1 \wedge B_1) \vee (A_1 \vee B_1) \wedge C_0$
$S_2 = A_2 \text{ XOR } B_2 \text{ XOR } C_1$ $\quad C_2 = (A_2 \wedge B_2) \vee (A_2 \vee B_2) \wedge C_1$
$S_3 = A_3 \text{ XOR } B_3 \text{ XOR } C_2$ $\quad C_3 = (A_3 \wedge B_3) \vee (A_3 \vee B_3) \wedge C_2$

Wir führen zwei Hilfsvariablen g_n und p_n ein:
$g_n = A_n \wedge B_n$ $\qquad p_n = A_n \vee B_n$

- g_n heißt Carry **generate**, weil ein Übertrag C_n gebildet wird, wenn sowohl A_n als auch B_n den Binärzustand 1 haben.
- p_n heißt Carry **propagate**, weil der Übertrag C_{n-1} weitergeleitet wird, wenn $p_n = 1$ und $g_n = 0$ ist.

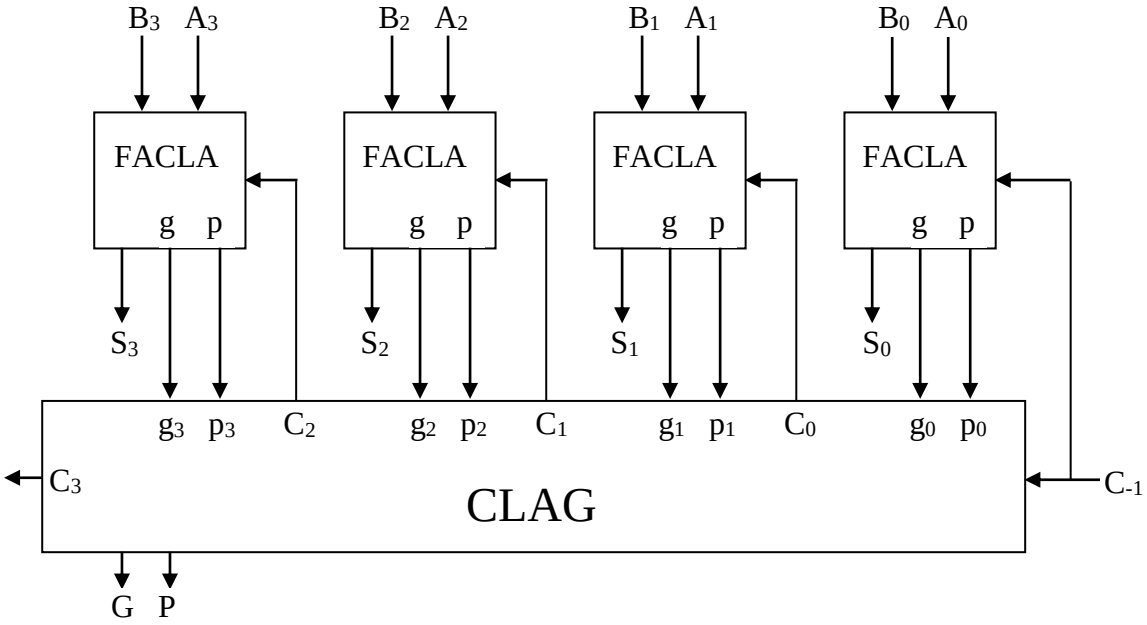
Setzt man g_n und p_n ein ergibt sich für die C_n: (ohne Klammerung bindet $\wedge$ stärker als $\vee$)

$C_0 = g_0 \vee p_0 \wedge C_{-1}$
$C_1 = g_1 \vee p_1 \wedge C_0$
$C_2 = g_2 \vee p_2 \wedge C_1$
$C_3 = g_3 \vee p_3 \wedge C_2$

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

$C_0 = g_0 \vee p_0 \wedge C_{-1}$
$C_1 = g_1 \vee p_1 \wedge g_0 \vee p_1 \wedge p_0 \wedge C_{-1}$
$C_2 = g_2 \vee p_2 \wedge g_1 \vee p_2 \wedge p_1 \wedge g_0 \vee p_2 \wedge p_1 \wedge p_0 \wedge C_{-1}$
$C_3 = g_3 \vee p_3 \wedge g_2 \vee p_3 \wedge p_2 \wedge g_1 \vee p_3 \wedge p_2 \wedge p_1 \wedge g_0 \vee p_3 \wedge p_2 \wedge p_1 \wedge p_0 \wedge C_{-1}$

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



Die Blackboxes FACLA (Full-Adder-Carry-Look-Ahead) erhalten die umgebaute Volladdiererschaltung. Der CLAG (Carry-Look-Ahead-Generator) erzeugt aus den g- und p-Hilfsvariablen die Überträge C_n. Man beachte: Die in dieser Schaltung benutzten Volladdierer in den FACLAs erzeugen keine Überträge.

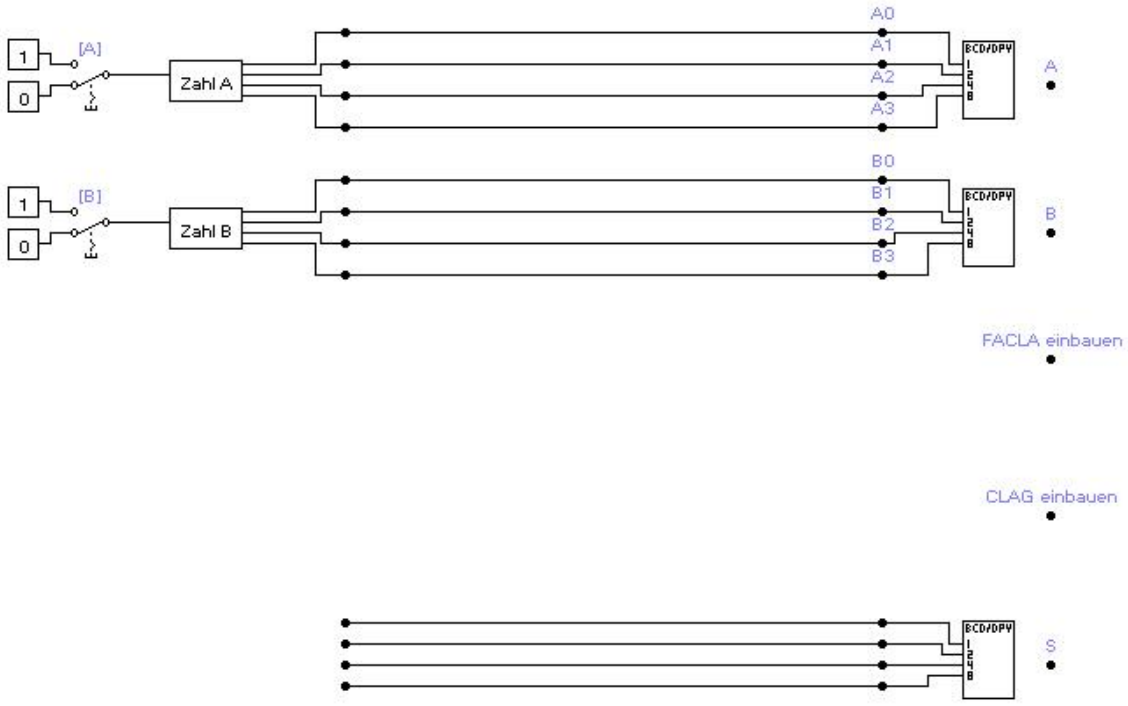
Über die Ein/Ausgänge C_{-1} C3, G und P können mit mehreren CLAGs mehrstufige Carry-Look-Ahead-Generatoren erzeugt werden:
$C_3 = g_3 \vee p_3 \wedge g_2 \vee p_3 \wedge p_2 \wedge g_1 \vee p_3 \wedge p_2 \wedge p_1 \wedge g_0 \vee p_3 \wedge p_2 \wedge p_1 \wedge p_0 \wedge C_{-1}$ mit
$G = g_3 \vee p_3 \wedge g_2 \vee p_3 \wedge p_2 \wedge g_1 \vee p_3 \wedge p_2 \wedge p_1 \wedge g_0$ und
$P = p_3 \wedge p_2 \wedge p_1 \wedge p_0$.
Eine Kaskadierung mit C_3,G, P sowie C_{-1} wird bei unseren Versuchen nicht benötigt. Daher ist $C_{-1} = 0$.

Vier Rechenbeispiele:

A	0001	0001	0010	0010	$a_3\ a_2\ a_1\ a_0$
B	0001	0011	0011	0111	$b_3\ b_2\ b_1\ b_0$
g = A ∧ B	000**1**	0001	00**1**0	0010	$g_3\ g_2\ g_1\ g_0$
p = A ∨ B	0001	00**1**1	0011	0**1**11	$p_3\ p_2\ p_1\ p_0$
$C_n = g_n \vee p_n \wedge C_{n-1}$	00**1**0	0**11**0	0**1**00	**11**00	$C_2\ C_1\ C_0\ C_{-1}$
S = A XOR B XOR C	0010	0100	0101	1001	$S_3\ S_2\ S_1\ S_0$

Versuch 630 Carry-Look-Ahead Addierwerk

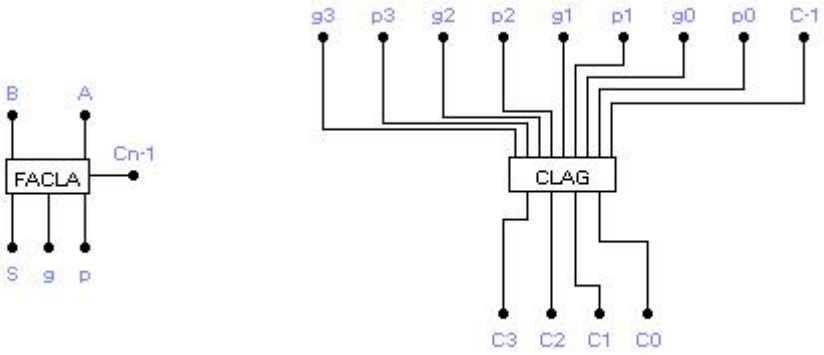
Bauen Sie in der Datei v630 ein vierstelliges Carry-Look-Ahead Addierwerk auf. Es werden vier FACLA und ein CLAG benötigt.



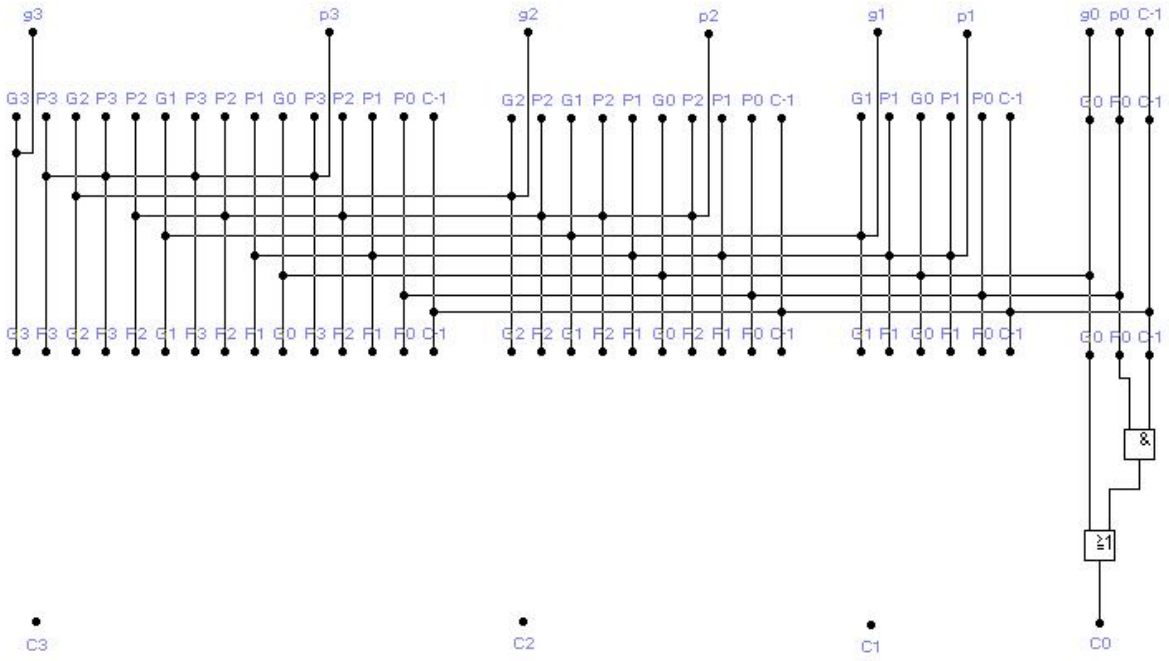
Vorgehensweise:

- Bauen Sie vier Makros FACLA ein. (im EWB bei Makros nachgucken)
- Bauen Sie ein Makro CLAG ein und vervollständigen Sie es. (Bauteile drehen mit Strg-R)
- Schließen Sie die FACLAs und den CLAG an die Summanden A und B und an die Summe S an.
- Überprüfen Sie die Schaltung auf korrekte Funktionsfähigkeit.

Anbei der Anschlussplan vom FACLA und CLAG:

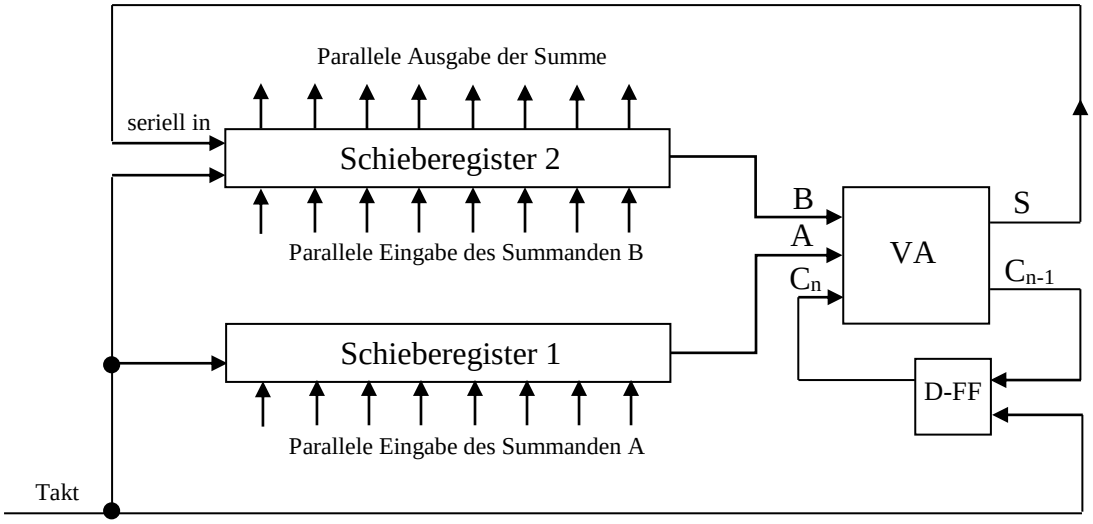


Anbei der Anschlussplan des CLAG:

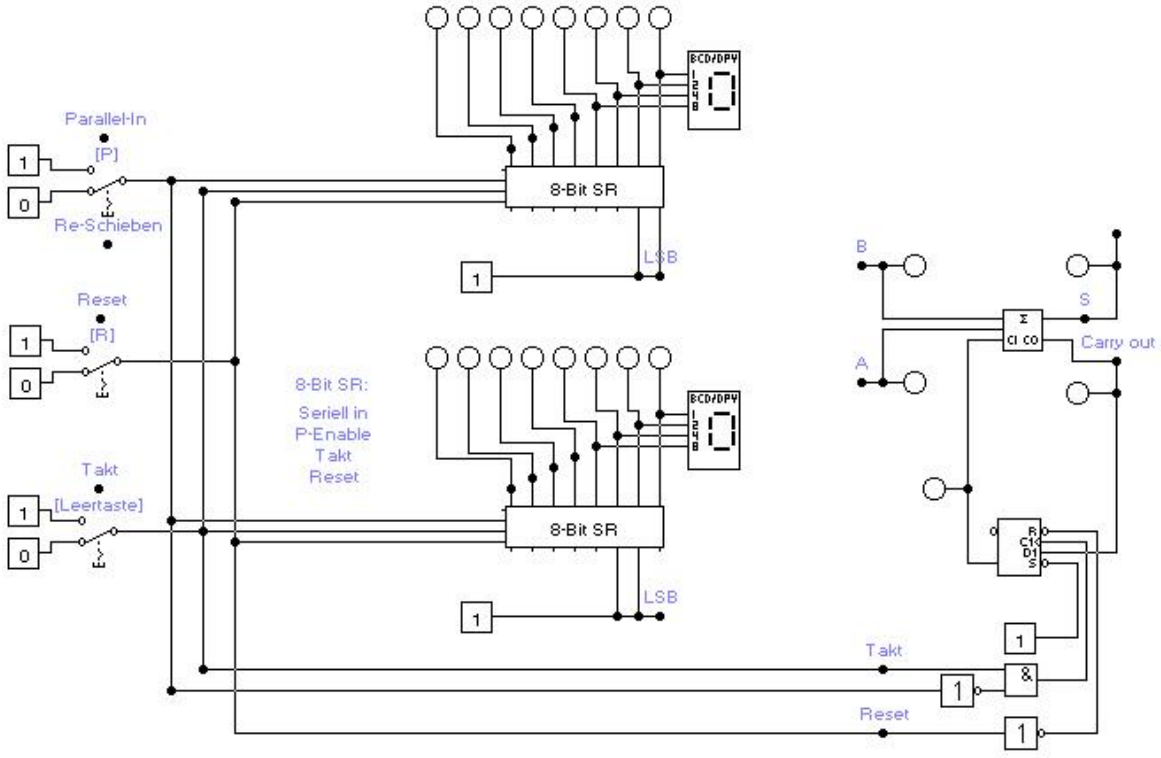


Versuch 640 Serielles Addierwerk

In diesem Versuch soll ein seriell arbeitendes Addierwerk untersucht werden, dessen Arbeitsprinzip die nächste Abbildung darstellt. Den beiden Schieberegistern 1 und 2 werden über parallele Eingaben die Summanden A und B eingegeben. Es gibt nur einen Volladdierer VA. 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.



In der Datei v640 finden Sie ein mit zwei Schieberegistern „8-Bit SR“ aufgebautes serielles Addierwerk:



Aufgabe:
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 0000wxyz testen, d.h. mit Summanden, deren ersten vier Stellen auf 0 gesetzt sind. Die Ausgabe erfolgt am oberen Schieberegister.

Bedienung:
- Löschen Sie alle SR-Register und das D-FlipFlop mit einem Reset.
- Laden Sie die Operanden, die an die unteren Paralleleingänge angelegt sind (3 oben und 6 unten), in die 8-Bit-SR-Register (Parallel-In).
- Schalten Sie das Addierwerk in den Rechts-Schiebemodus.
- Führen Sie eine 8-Bit Addition durch (8 x takten) und überprüfen Sie das Ergebnis auf seine Richtigkeit.

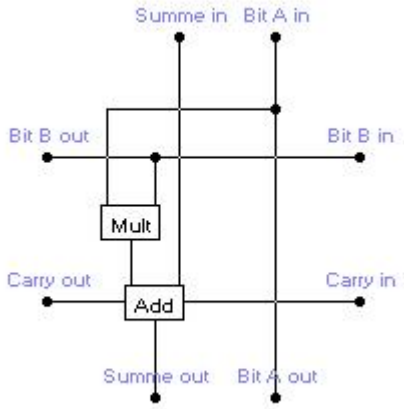
Versuch 650 Multiplikation

In Versuch 650 soll die Multiplikation, die nach dem bürgerlichen Multiplikationsalgorithmus arbeitet, untersucht werden. Dazu ist ein Beispiel einer Multiplikation nach diesem Verfahren mit den Faktoren A=1101 und B=1011 gegeben:

```
1)  1101 * 1011     anders sortiert 2)  1101 * 1011
       1101                                 1101
        0000                               1101
         1101                             0000
          1101                          1101
       10001111                         10001111
```

Das Produkt der Faktoren A*B wird durch das Produkt S=10001111 dargestellt.

Sie sollen in diesem Versuch ein vierstelliges Multiplikationswerk nach der obigen Darstellung 2) für das duale Ziffernsystem {0,1} entwickeln. Dafür brauchen wir ein EWB-Makro „X“ mit vier Ein- und vier Ausgängen:

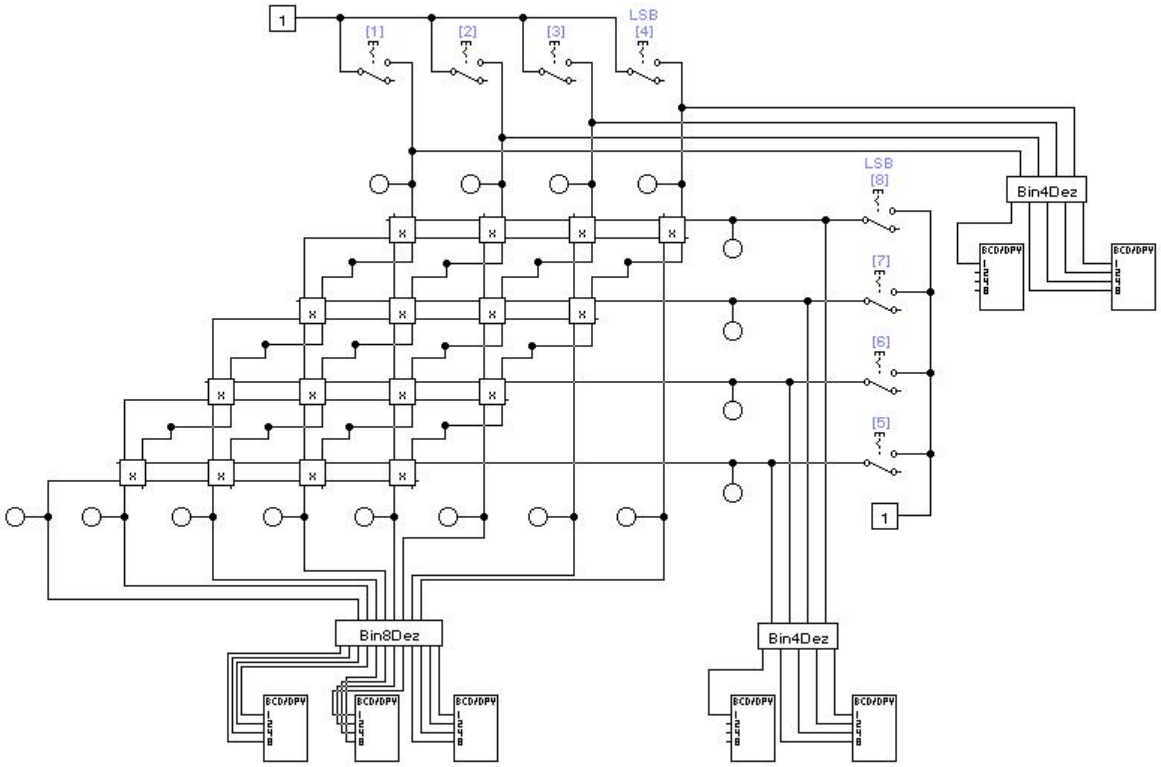


- Das EWB-Makro „Mult“ muss in diesem Fall das Einmaleins beherrschen:
  $0 \cdot 0 = 0$
  $0 \cdot 1 = 0$
  $1 \cdot 0 = 0$
  $1 \cdot 1 = 1$
  Welches Gatter berechnet diese Funktion?

- Das EWB-Makro „Add“ soll einen fertigen Addiererbaustein aus der Bauteilbibliothek des EWB enthalten.

- Entwickeln Sie die obige Schaltung und bauen Sie sie in der Datei v670 in das EWB-Makro „X“ ein. Wenn Sie alles richtig gemacht haben, kann die Schaltung multiplizieren.

  Hinweis:
  Sie können anstelle der Makros Add und Mult auch die nötigen Bauteile direkt in das Makro X einbauen. Das erspart etwas Arbeit.



Die EWB-Makros „Bin4-Dez“ und „Bin8-Dez“ enthalten Wandler, die Binärzahlen in BCD-Zahlen umformen, die von den Sieben-Segment-Anzeigen als Dezimalzahlen angezeigt werden. Sie können also die Faktoren A = a_3 a_2 a_1 a_0 und B = b_3 b_2 b_1 b_0 mit den Tasten als Binärzahlen eingeben und als Dezimalzahlen an den Sieben-Segment-Anzeigen ablesen. Auch das berechnete Produkt ist als Dezimalzahl ablesbar.