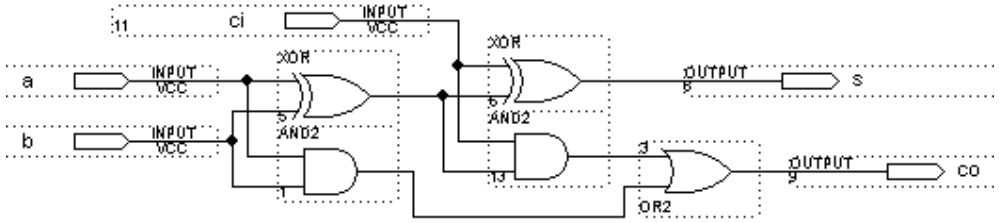
Lösung Versuch Nr. 3

1: 4-Bit Addierer

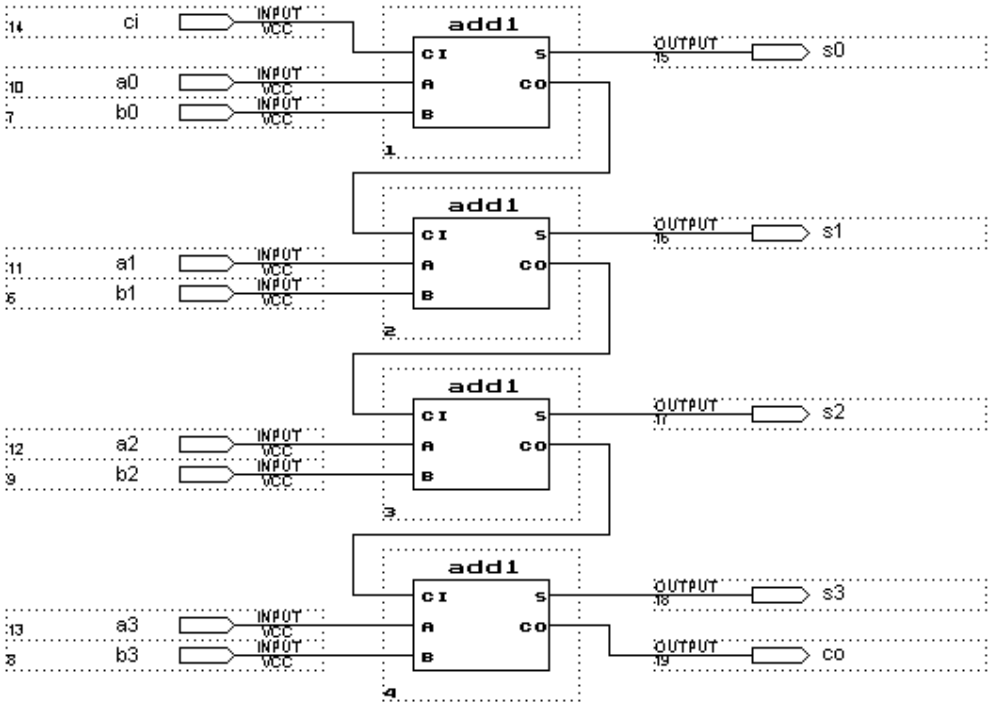
Aufgaben

1. *Zeichnen Sie das Schaltschema eines 1-Bit (Voll) Addierers mit den Eingängen a, b, ci (carry input) und den Ausgängen s und co (carry output). Erzeugen Sie davon ein Symbol für die weitere Verwendung.*



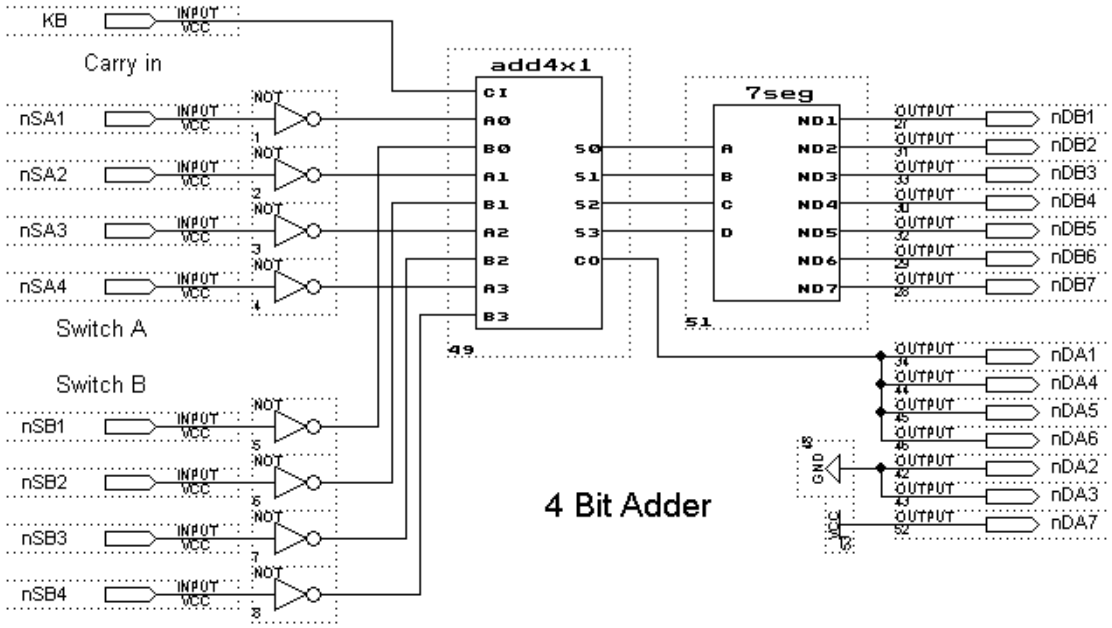
1-Bit Volladdierer

2. *Setzen Sie wieder in einem neuen Grafikfile (add4x1.gdf) einen 4-Bit Addierer (ripple Carry) aus 4 1-Bit Addierstufen (add1) zusammen. Erzeugen Sie davon wieder ein Symbol für die weitere Verwendung.*



4-Bit Ripple-Carry-Addierer

3. *Um den Addierer austesten zu können, wollen wir die beiden Summanden mit den beiden 4-Bit Schaltern (SA und SB) auf dem Board eingeben und die Summe auf die 7-Segment Anzeige ausgeben.*



4-Bit Addierer mit Schalter Eingabe und Resultat Ausgabe auf 7-Segment Anzeige

2: 2er Komplement Darstellung

Aufgaben

5. *Wir wollen jetzt experimentell ein paar dieser Bitkombinationen bestimmen.*

Dez	Bin	Hex		Dez	Bin
-1	1111	F		1	0001
-2	1110	E		2	0010
-3	1101	D		3	0011
-4	1100	C		4	0100

7. *Finden sie durch Vergleich der Bitkombinationen für +1 und -1, +2 und -2, +3 mit -3 die einfache Regel zur Bildung des 2er Komplements (-1 ist das 2er Komplement von +1 und umgekehrt).*

   Alle Bits invertieren, danach 1 addieren
   (gilt für die Konversion in beiden Richtungen.

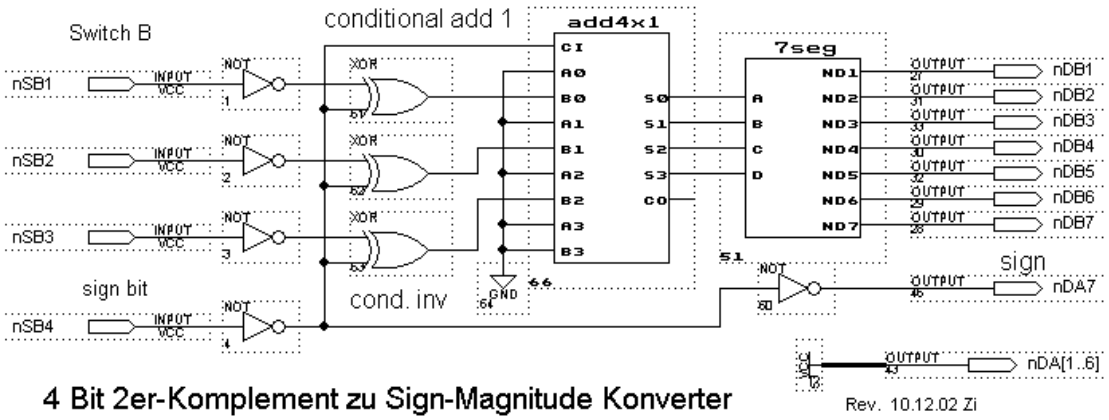
8. *Das höchstwertige Bit (MSB = Most Significant Bit) enthält die Information über das Vorzeichen. Tragen Sie die richtige Zuordnung auf dem Lösungsblatt ein.*

   positiv: MSB = 0 negativ: MSB = 1

9. *Entwickeln Sie nach der unter 7. gefundenen Regel eine Schaltung zur Konversion von 4-Bit 2er Komplement Zahlen in Betrag und Vorzeichen.*

Bei einer negativen Zahl (MSB=1) müssen alle Bits invertiert und dazu 1 addiert werden. Bei einer positiven Zahl (MSB=0) können alle Bits direkt (ohne Inversion und 1 Addition) zur Anzeige auf den 7-Segment Decoder durchgeschaltet werden.

Wir brauchen somit wahlweise (gesteuert durch das MSB) eine Inversion oder Nichtinversion vor dem Addierer und müssen entweder 1 oder 0 addieren. Die programmierbare Inversion macht man am einfachsten mit XOR-Toren, die Addition von 1 oder 0 über den Carryeingang des Addierers.



4-Bit 2er Komplement zu Sign-Magnitude Konversion

Achtung: Eigentlich müssen alle Bits (einschliesslich Vorzeichen!) invertiert und dann 1 dazu addiert werden. In unserem Fall bleibt das MSB jedoch immer 0 (positiv wird nicht invertiert, negativ wird von 1 nach 0 invertiert). Deshalb kann auf die programmierte Inversion und damit das 4. XOR-Tor verzichtet werden. Ebenso kann der Addierer für das höchstwertige Bit 3 eingespart werden, S3 wird gleich dem Carry Ausgang co von Bit 2.

Alternative Lösung mit Tabelle:

```
% V3  2sComplement-SignMagnitude Converter %
% 7.12.02                                  %
%                                 %
SUBDESIGN 2sC_SigMag_btab
(   C3, C2, C1, C0        : INPUT;
    nSig,  M3, M2, M1, M0 : OUTPUT; )
BEGIN
 % Look Up Table 2sComplement - SignMagnitude %
TABLE
% 4bit 2s Compl.        sign - magnitude output %
  C3, C2, C1, C0    =>  nSig,   M3, M2, M1, M0;
   0,  0,  0,  0    =>    1,    0,  0,  0,  0; % +0 %
   0,  0,  0,  1    =>    1,    0,  0,  0,  1; % +1 %
   0,  0,  1,  0    =>    1,    0,  0,  1,  0; % +2 %
   0,  0,  1,  1    =>    1,    0,  0,  1,  1; % +3 %
   0,  1,  0,  0    =>    1,    0,  1,  0,  0; % +4 %
   0,  1,  0,  1    =>    1,    0,  1,  0,  1; % +5 %
   0,  1,  1,  0    =>    1,    0,  1,  1,  0; % +6 %
   0,  1,  1,  1    =>    1,    0,  1,  1,  1; % +7 %
   1,  0,  0,  0    =>    0,    1,  0,  0,  0; % -8 %
   1,  0,  0,  1    =>    0,    0,  1,  1,  1; % -7 %
```

```
   1,  0,  1,  0    =>    0,    0,  1,  1,  0; % -6 %
   1,  0,  1,  1    =>    0,    0,  1,  0,  1; % -5 %
   1,  1,  0,  0    =>    0,    0,  1,  0,  0; % -4 %
   1,  1,  0,  1    =>    0,    0,  0,  1,  1; % -3 %
   1,  1,  1,  0    =>    0,    0,  0,  1,  0; % -2 %
   1,  1,  1,  1    =>    0,    0,  0,  0,  1; % -1 %
END TABLE;
END;
```

10. Bestimmen Sie den Wertebereich (kleinste negative, grösste positive Zahl) für 4-Bit 2er Komplementzahlen.

- 8 bis +7

11. Die Reportfiles Add4.rpt. und 2sC-SM.rpt. enthalten am Anfang, direkt nach der Copyright-Predigt die Kurzinformationenfinden unter Device Summary / Utilization die Ausnützung des PLDs in %.

```
** DEVICE SUMMARY **
Chip/                 Input  Output  Bidir      Shareable
POF       Device      Pins   Pins    Pins  LCs  Expanders  % Utilized
4-Bit_Adder
      EPM3064ALC44-10    9      14      0   26      14          40 %
User Pins:               9      14      0
```

Device Summary aus Report file (rpt)

In der Anzeige A muss nur 0 oder 1 dargestellt werden. Dies kann auf die gleiche Art wie im Versuch 2 oder mit einem zweiten 7-Segment Decoder gemacht werden. Die Compilation führt in beiden Fällen zum exakt gleichen Resultat.

```
** DEVICE SUMMARY **
Chip/                 Input  Output  Bidir      Shareable
POF       Device      Pins   Pins    Pins  LCs  Expanders  % Utilized
4-Bit_2er-Kompl_to_sign-mag
      EPM3064ALC44-10   4      14      0   14      0          21 %
User Pins:              4      14      0
```

Device Summary aus 2sC-SM.rpt

Hier ergibt sich für im Gegensatz zu vorher kein Unterschied in der Ausnützung für die beiden verschiedenen Realisierungen des 7-Segment Decoders. Die im Report File dokumentierten logischen Gleichungen sind in beiden Fällen weitgehend identisch (leicht unterschiedliche Minimisierugen).

12. Wie erklären Sie sich den unerwarteten Unterschied in der Belegung der Chip-Resourcen?

Der Compiler implementiert nicht die Schaltung sondern deren Funktion.

In Aufgabe 7 (2er Komplement Darstellung) degeneriert der Addierer zu X+0 und X+1 (gegenüber X+Y) in der Aufgabe 3. Dadurch kann er mit nur Halbaddierern realisiert werden anstatt mit Volladdierern und die Halbierung des Aufwandes ist damit bereits qualitativ erklärt.