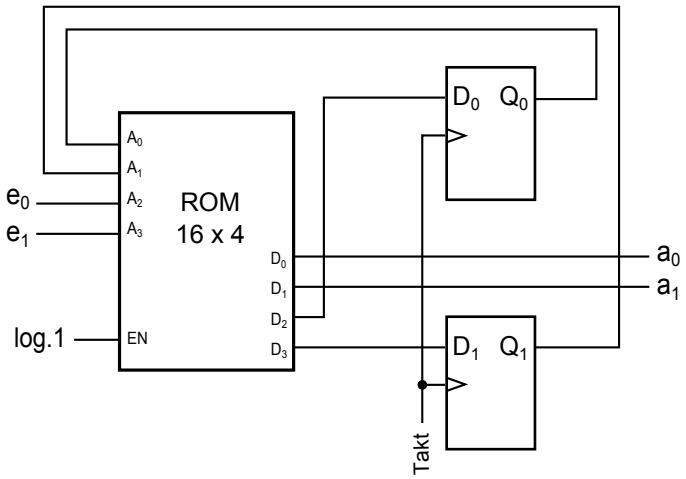
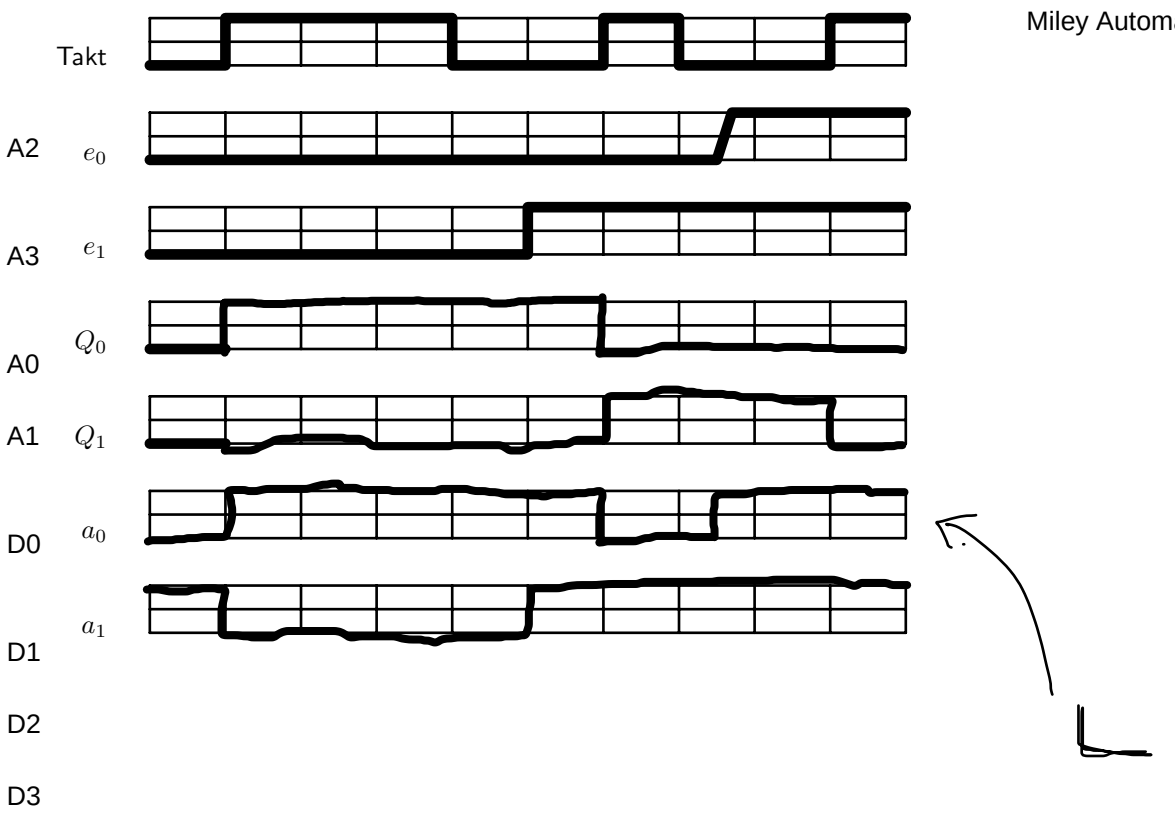
Aufgabe 1: Timing Diagramm

Gegeben ist das dargestellte Schaltwerk mit zwei Eingabesignalen e_0 und e_1 und zwei Ausgabesignalen a_0 und a_1. Im 16×4 ROM-Baustein, der Teil der Schaltung ist, sind die in der Tabelle angegebenen Werte gespeichert:

A_3	A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1	1	0
0	0	0	1	1	0	0	1
0	0	1	0	1	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	1	1	0	0
0	1	1	0	1	0	1	1
0	1	1	1	0	0	0	0
1	0	0	0	0	0	1	1
1	0	0	1	1	0	1	1
1	0	1	0	1	1	1	0
1	0	1	1	1	0	1	1
1	1	0	0	0	1	1	0
1	1	0	1	1	0	0	1
1	1	1	0	0	0	1	1
1	1	1	1	1	1	0	0



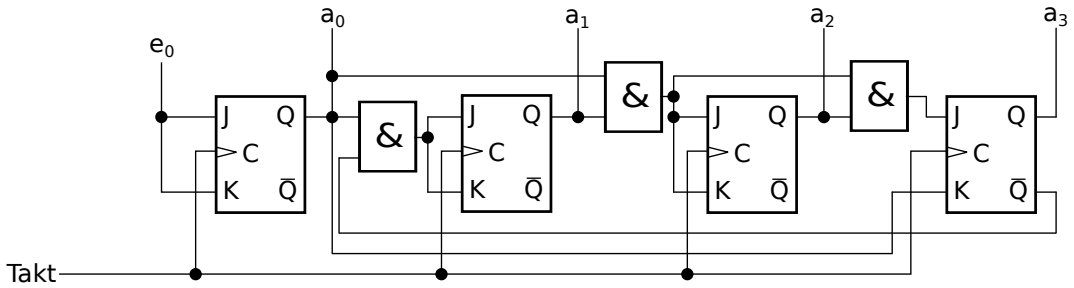
Zeichnen Sie den Verlauf der Signale Q_0, Q_1, a_0 und a_1.



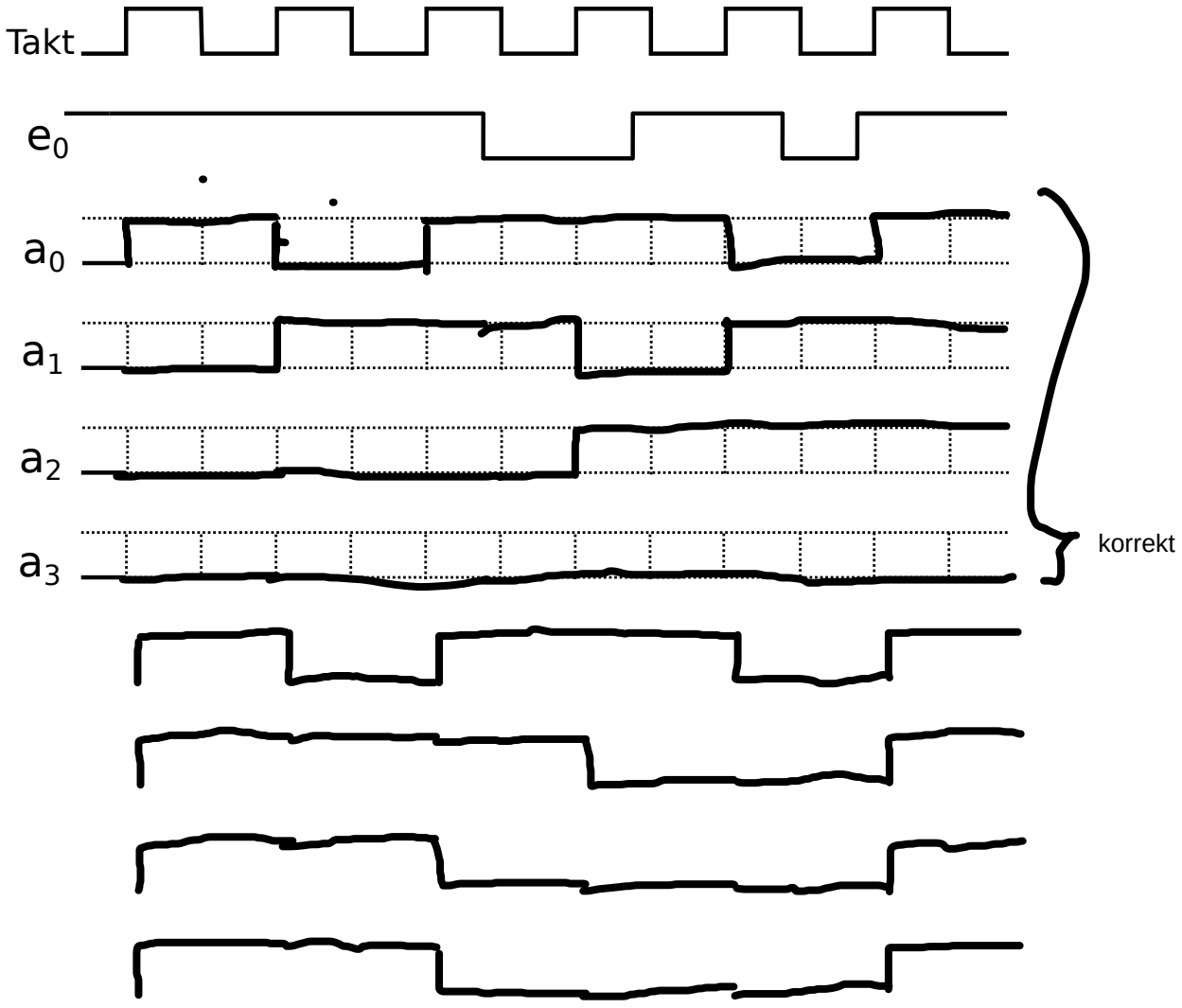
Aufgabe 2: Timing-Diagramm

Das Auswerten der JK-Flipflops passiert simultan, sprich es werden die alten Outputs betrachtet.

Es ist folgende Schaltung gegeben:

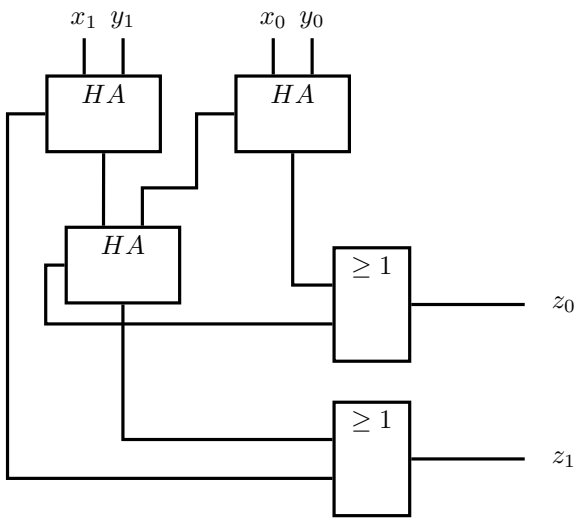


Überlegen Sie sich die Funktionsweise der Schaltung, sodass Sie diese in der Übung erklären können und vervollständigen Sie das nachfolgende Timing-Diagramm. Achten Sie dabei auf die Flankentriggerung.



Aufgabe 3: Schaltung zu ROM

Gegeben ist folgendes Schaltbild, das aus Halbaddierer-Bausteinen sowie einem Gatter besteht. Der linke Ausgang des HA ist C_{out}.

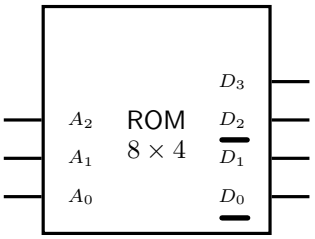


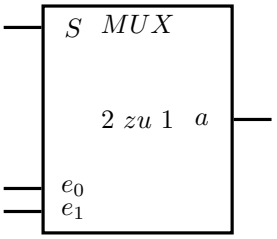
a) Analysieren Sie die oben gegebene Schaltung und vervollständigen Sie den Vordruck der Wahrheitstabelle.

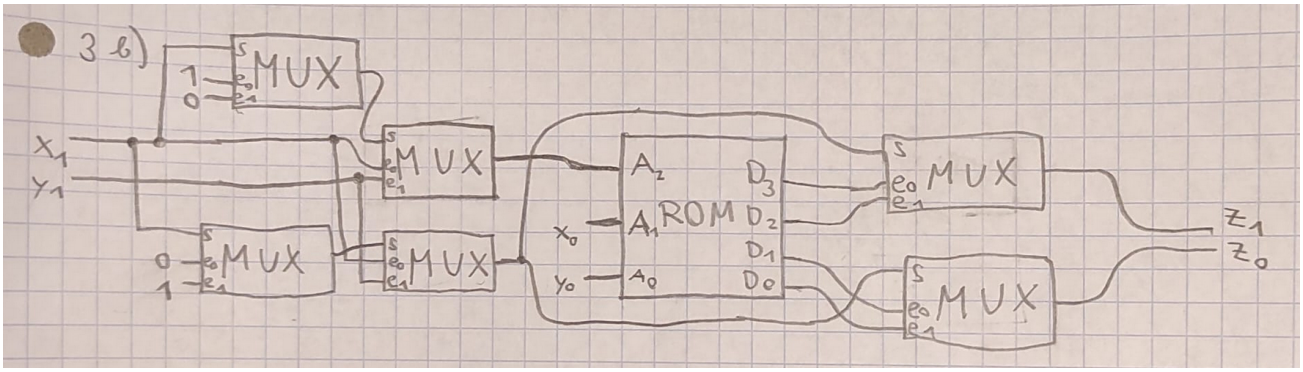
x_1	y_1	x_0	y_0	z_0	z_1
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	0	1

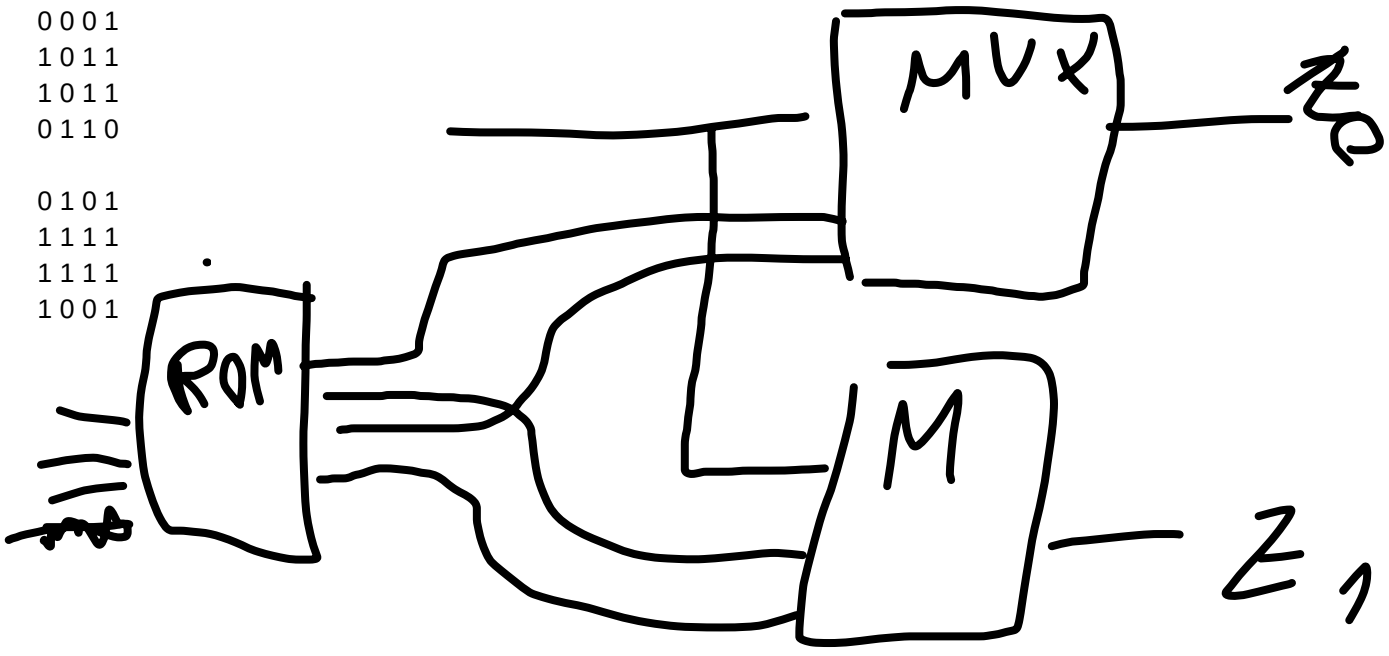
b) Realisieren Sie die Wahrheitstabelle aus Aufgabe a) mit den unten gegebenen Bausteinen (Sie können den Multiplexer mehrmals verwenden, den ROM jedoch nur einmal). Geben Sie außerdem den Inhalt des ROMs an.

A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	0	1	0
0	1	1	1	0	0	0
1	0	0	1	1	0	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	0









Aufgabe 4: Entwurf & Realisierung einer Boole'schen Funktion via PLA

Entwerfen Sie einen 2×2-*Bit Addierer* mit den vier Eingängen a, b, c und d, der die Summe

$$(ab)_2 + (cd)_2 = (s_2 s_1 s_0)_2$$

auf die drei Ausgänge s_2, s_1 und s_0 abbildet. Um auch negative Zahlen verarbeiten zu können sollen die Eingabebits als Zweierkomplement interpretiert werden. Beachten Sie, dass die Darstellung der negativen Zahlen im 2-Bit Zweierkomplement und im 3-Bit Zweierkomplement nicht ident ist!

a) Vervollständigen Sie die nachfolgende Wahrheitstabelle. Bestimmen Sie anschließend jene minimale Normalform, die Sie für Unteraufgabe b) benötigen. Sie können für die Bestimmung der Normalform beliebige Werkzeuge verwenden. Wir empfehlen den an der Uni Marburg entwickelten Karnaugh-Veitch Map Rechner[1]. Sie müssen nur die Eigenschaften des Ergebnisses (Normalform) und nicht die Funktionsweise des verwendeten Werkzeuges erklären können.

a	b	c	d	s_2	s_1	s_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	1	1	0
0	0	1	1	1	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0
1	0	0	0	1	1	0
1	0	0	1	1	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	1	1	1
1	1	0	1	0	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

000
000

000

001
110

111

$s_2 =$ (¬b∧c)∨(c∧¬d)∨(a∧¬b)∨(a∧¬d)∨(a∧c)

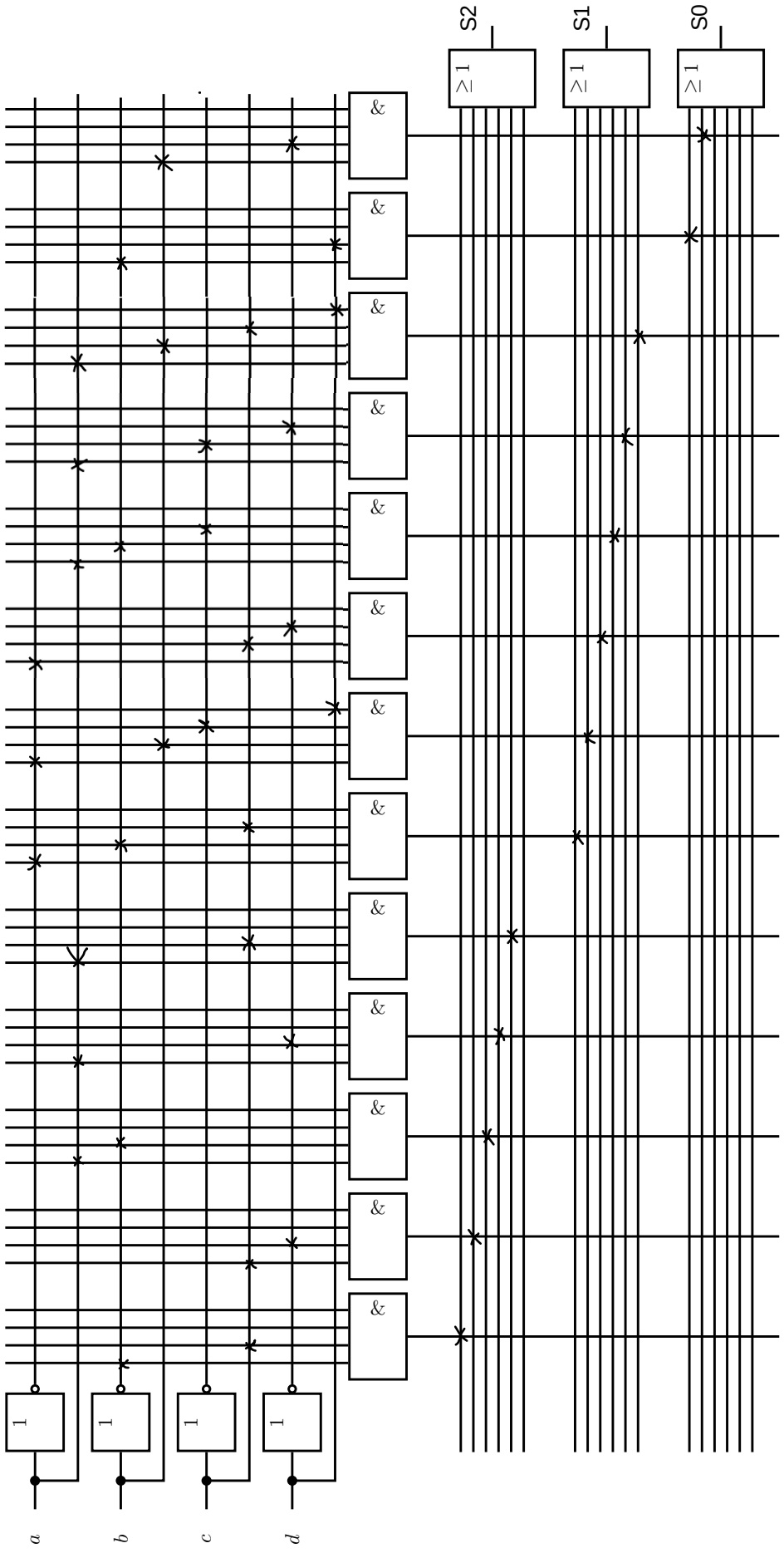
$s_1 =$ (¬a∧¬b∧c)∨(¬a∧b∧¬c∧d)∨(¬a∧c∧¬d)∨ (a∧¬b∧¬c)∨(a∧¬c∧¬d)∨(a∧b∧c∧d)

$s_0 =$ (¬b∧d)∨(b∧¬d)

[1] https://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/code/karnaughmap/

b) Realisieren Sie den 2×2-Bit Addierer im unten dargestellten PLA. Vergessen Sie nicht Ein- und Ausgänge zu beschriften.

restlichen Verbindungen müssen auch eingezeichnet werden

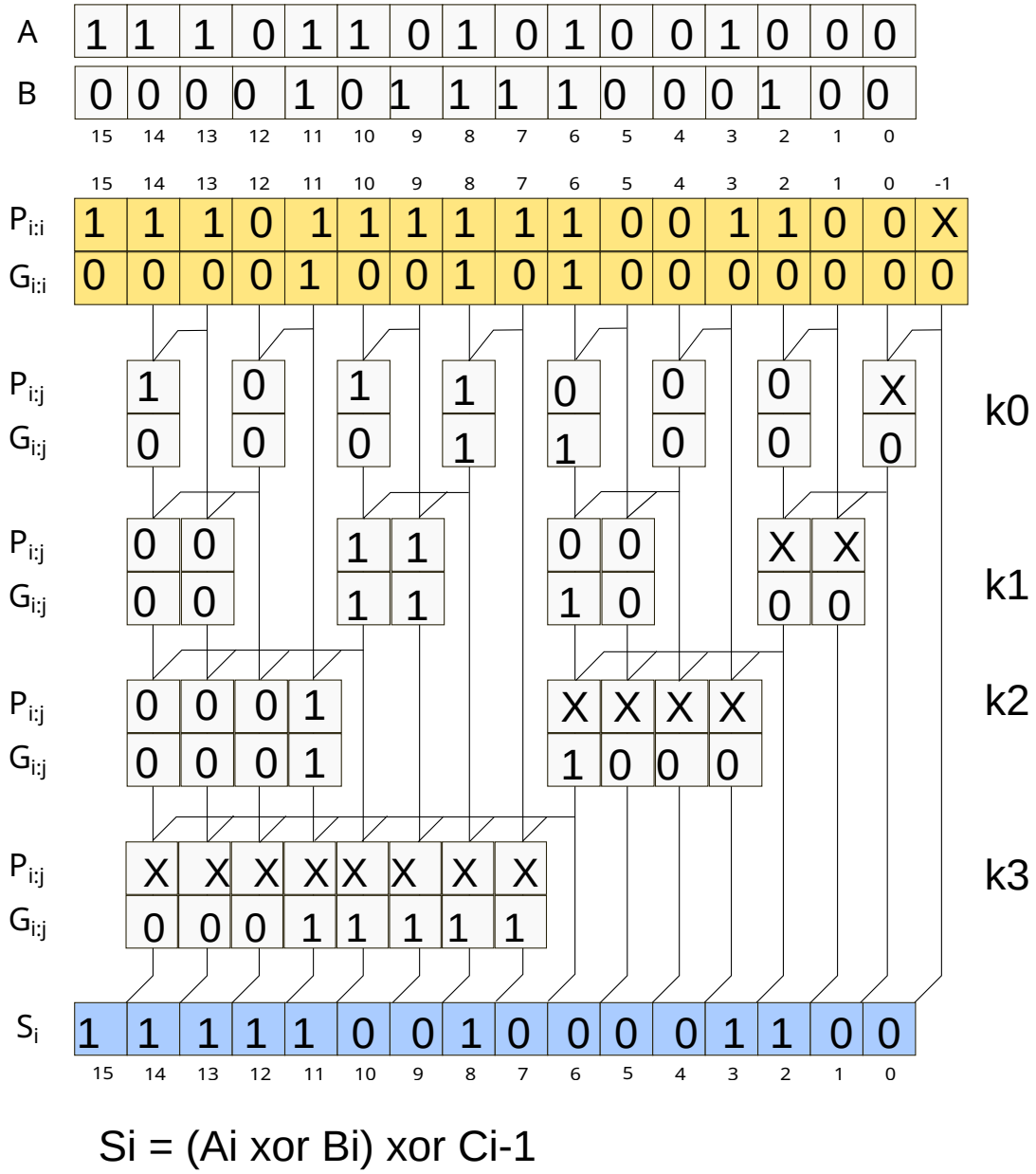


Aufgabe 5: Prefix-Addierer

a) Erklären Sie, wie ein Prefix-Addierer die Addition implementiert. Gehen Sie dabei auf die G bzw. P Signale ein.

Ein Prefix-Addierer berechnet zuerst die G- und P-Signale für jede Reihe der Eingabedaten.
Ein G-Signal auf 1 bedeutet an dieser Stelle wird ein Übertrag generiert (Gi = Ai AND Bi).
Ein P-Signal auf 1 bedeutet an dieser Stelle kann ein Übertrag propagiert werden (Pi = Ai OR Bi).
Danach werden schrittweise (2er Potenzen) die Reihen zu immer größeren "G-P-Blöcken" zusammengefasst (zuerst 2 zu einem, dann 4, dann 8 usw.) für welche zudem die G- und P-Signale berechnet werden (Gi:j = Gi:k + Pi:kGk-1:j, Pi:j = Pi:kPk-1:j). Dies wird solange gemacht, bis man die Größe der Eingabe Daten erreicht hat. Die Summe kann somit stellenweise berechnet werden durch die Formel Si = (Ai xor Bi) xor Ci-1, wobei Ci-1 = Gi-1:-1. (Gi-1:-1 sind hierbei die Prefixes)

b) Gegeben ist die folgende Skizze eines Prefix-Addierers. Berechnen Sie $(-4792)_{10} + (3012)_{10}$ indem Sie die untenstehende Skizze mit den konkreten Werten befüllen. Nehmen Sie $C_{in} = 0$ an.



Aufgabe 6: Geschwindigkeit von Addierern

Wie Sie in der Vorlesung gehört haben, gibt es verschiedene Implementierungen von Addierern. In dieser Übung werden Sie sich die Geschwindigkeit (Latenz) von den vorgestellten Addierern näher ansehen.

a) Bevor Sie die Addierer vergleichen können, sollten Sie die Formeln für die Latenz der Addierer verstehen. Unten finden Sie die in der Vorlesung präsentierten Formeln. Erklären Sie diese für jeden Addierer. Falls notwendig, zeichnen Sie sich Skizzen der Schaltungen, um die Formeln besser erklären zu können.

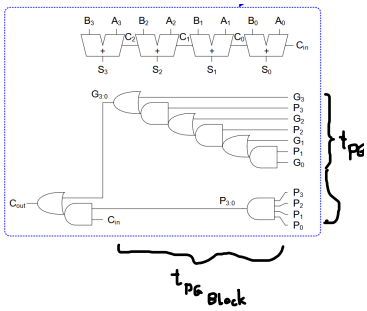
- Ripple-Carry-Addierer: $t_{ripple} = Nt_{FA}$

  N ist die Anzahl der Bits, die addiert werden und tFA die Zeit, welche ein Full-Adder (1Bit)

  braucht. Beim RC-Addierer werden die Full-Adder einfach seriell hintereinander geschaltenund somit N mal die Zeit des Full-Adders benötigt.

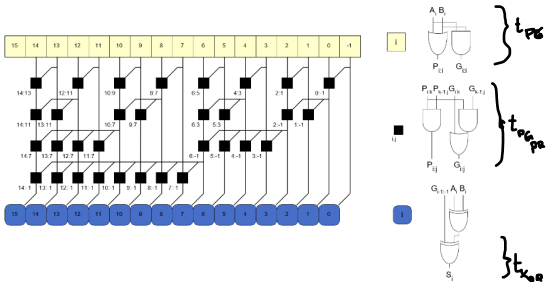
- Carry-Lookahead-Addierer: $t_{CLA} = t_{PG} + t_{PG_{Block}} + (\frac{N}{k} - 1)t_{and\&or} + kt_{FA}$

  tPG ist die Zeit, welche benötigt wird um die P- und G-Signale für die einzelnen Reihen
  zu berechnen (also für eine Reihe, da dies parallel geschieht).tPGBlock ist die Zeit, welche benötigt wird um die P- und G-Signale für die ganzen k-Blöcke zu berechnen (also für einen, da dies parallel geschieht).
  Wir benötigen N/k - 1 Mal die Zeit von tand&or (Zeit für das Berechnen des Cout), da wir für den letzten Block kein Cout mehr berechnen müssen, hierbei darf nämlich kein Übertrag mehr passieren (Overflow).
  ktFA steht für die Zeit, welche k FA brauchen, wenn sie hintereinander ausgeführt werden (Finale Berechnung).

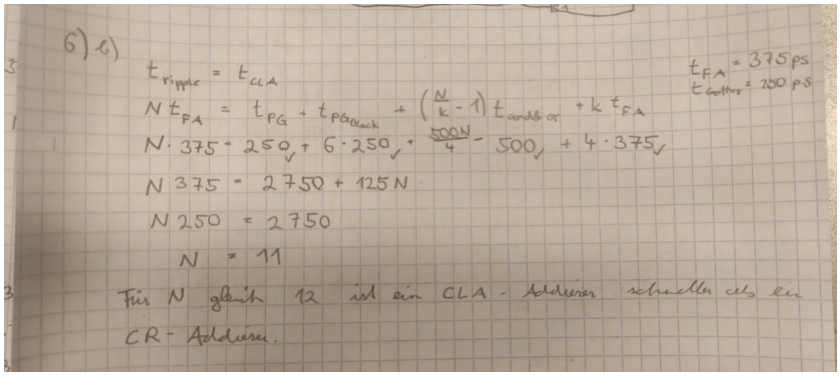


- Prefix-Addierer: $t_{PA} = t_{PG} + log_2N(t_{PG_{Prefix}}) + t_{xor}$
  Erklären Sie auch wieso der letzte Summand nicht $2t_{xor}$ ist.

  tPG ist der gleiche Faktor wie beim CL-Addierer. tPGPrefix ist die Zeit, welche benötigt wird um die G- und P-Signale eines "Blocks" (Prefixe) zu berechnen, diese werden mit log2N multipliziert, da man log2N "Ebenen" hat auf denen diese simultan berechnet werden. txor ist die Zeit die das xor-Gatter benötigt zum Berechnen der Summe, es wird hier nur ein txor benötigt, weil das erste xor-Gatter bereits während den anderen Berechnungen berechnet wird.



b) Nehmen Sie an, dass ein Volladdierer eine Latenz von $375ps$ hat. Weiters, nehmen Sie an, dass Gatter mit 2 Eingängen eine Latenz von $250ps$ haben. Berechnen Sie, ab welcher Anzahl an Bits (N) ein Carry-Lookahead-Addierer ($k = 4$) schneller ist, als ein Ripple-Carry-Addierer. Betrachten Sie nur N die durch k teilbar sind.

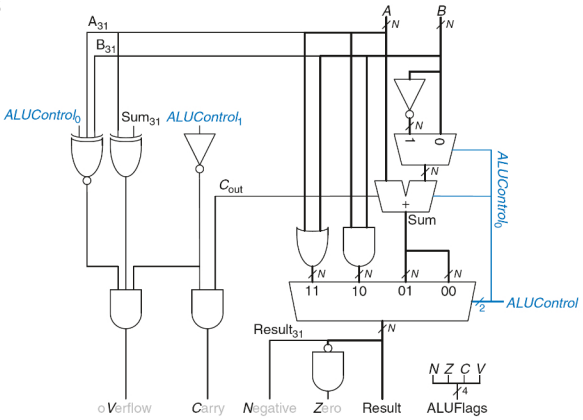


Aufgabe 7: Funktionen einer ALU

Diese Aufgabe bezieht sich auf die ALU aus der Vorlesung (siehe `03_building_blocks.pdf`, Folie 34 ff.). Seien Sie sich bewusst, dass Sie die Lösung anhand Ihrer Abgabe erklären müssen. Machen Sie daher Skizzen zum Aufbau der ALU, falls das dazu notwendig ist.

a) Erklären Sie wie die ALU eine Subtraktion umsetzt. Gehen Sie dabei auch auf die Notwendigkeit der Verbindung zwischen $\texttt{ALUControl}_0$ und dem Addierer ein.

Eine Subtraktion (A - B) bei der ALU ist eine Addition der beiden Zahlen A + ¬B + 1 (¬B + 1 ist das Zweierkomplement der Zahl B). Dies wird realisiert durch einen Adder, welchem die Zahl A und durch einen MUX das Komplement der Zahl B sowie eine 1 beim Carry-In-Kanal gefüttert wird. ALUControl0 ist das 0-Bit der ALUControl (Steuerung über MUX, welche Funktion durchgeschalten wird) und ist 1 wenn die Subtraktion durchgeführt werden soll (ALUControl für Subtraktion 01). Dieses Signal sorgt somit für die Steurung des MUXs für Bs Komplement und für das befüllen des Carry-In-Kanals mit 1.



b) Erklären Sie wie die ALU die vier Status Ausgänge (`V`, `C`, `N`, `Z`) umsetzt. Legen Sie den Fokus der Erklärung auf das oVerflow flag.

Z überprüft, ob das Resultat gleich 0 ist, indem es das Komplement des Resultats nimmt und dann verundet.
N überprüft, ob das Resultat negativ ist, dies geschieht indem das MSB betrachtet wird (0 = pos, 1 = neg).
C überprüft, ob ein Übertrag stattgefunden hat, indem es überprüft ob eine Subtraktion oder Addition statt= gefunden hat und ob Cout = 1 ist.
V überprüft, ob ein Overflow stattgefunden hat, hierfür werden drei Dinge relevant:

- Es muss eine Subtraktion oder Addition stattfinden.
- Die Zahlen A und B müssen ein anderes Vorzeichen als die Summe aufweisen.
- Es muss entweder eine Addition stattfinden, bei der die beiden Zahlen das gleiche Vorzeichen haben oder eine Subtraktion, bei der die Zahlen unterschiedliche Vorzeichen haben.

Aufgabe 8: ROM Erweiterung

Die untenstehenden Aufgaben fordern Sie dazu auf, ROM Bausteine aus anderen ROM Bausteinen zu konstruieren. Die drei untenstehenden Abbildungen zeigen alle möglichen Bausteine, die Sie brauchen. Beachten Sie, dass Sie **nur jene Bausteine verwenden dürfen, die in der Aufgabe beschrieben werden**.

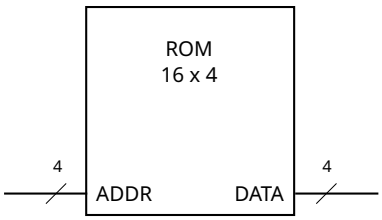


Abbildung 1: 16 x 4 ROM Baustein

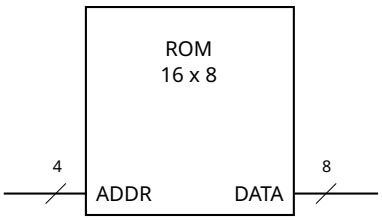


Abbildung 2: 16 x 8 ROM Baustein

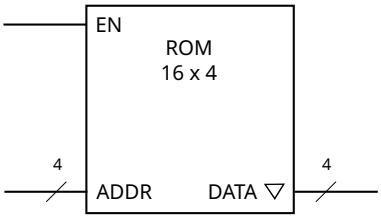
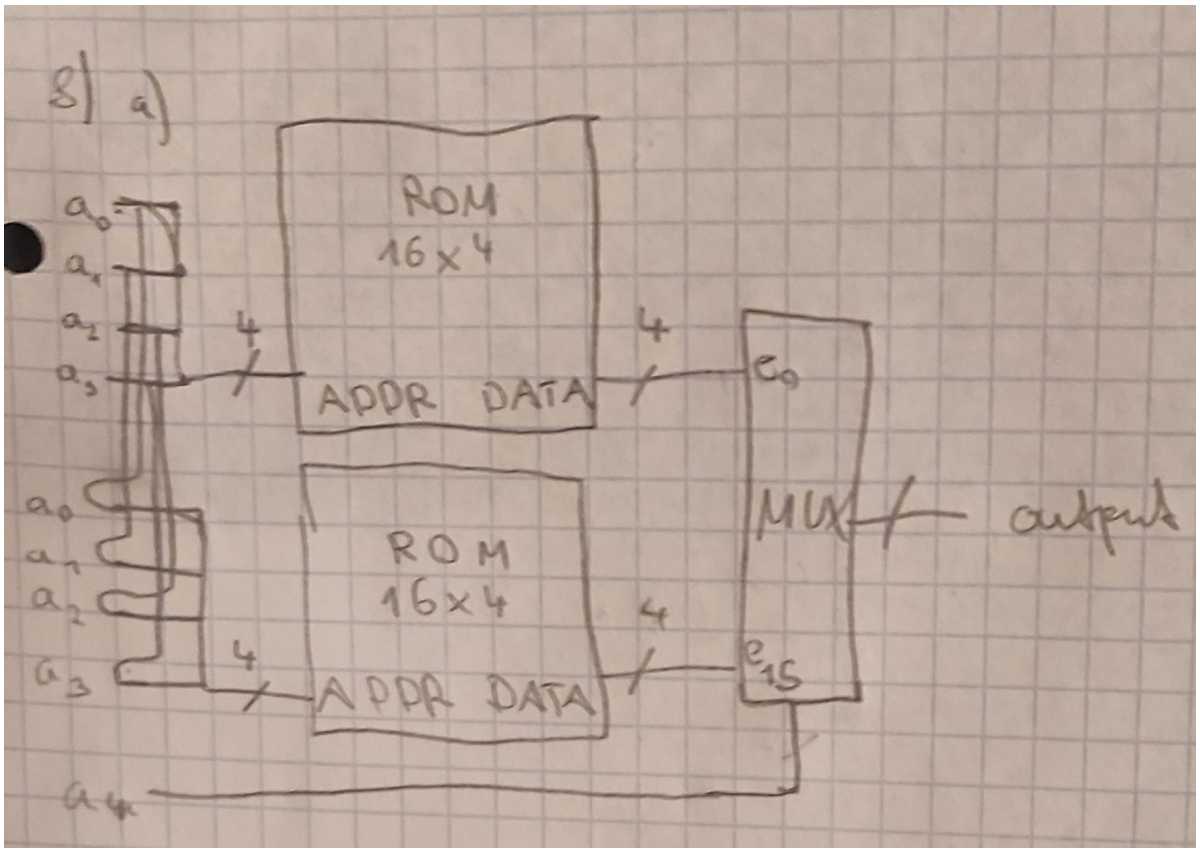
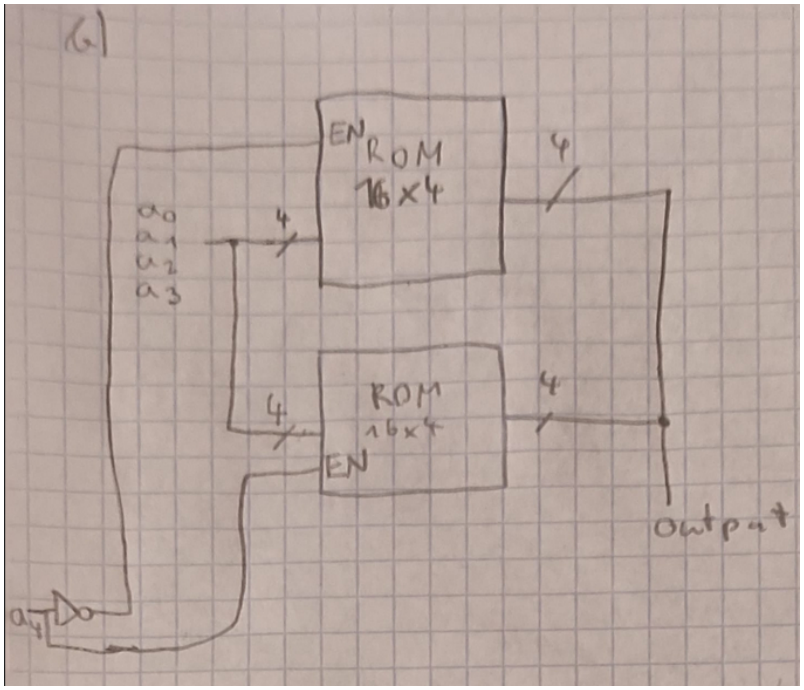


Abbildung 3: 16 x 4 ROM Baustein mit Enable-Eingang

a) Ihnen stehen zwei 16x4 ROM Bausteine zur Verfügung. Kombinieren Sie diese zu einem 32x4 ROM Baustein in dem Sie einen Multiplexer mit 4-bit breiten Datensignalen verwenden.



b) Ihnen stehen ein Inverter und zwei 16x4 ROM Bausteine mit enable Eingang und Tri-State Ausgang zur Verfügung. Kombinieren Sie diese zu einem 32x4 ROM Baustein, ohne weitere Schaltelemente zu verwenden.



c) Ihnen steht ein 16x8 ROM zur Verfügung. Wie können Sie diesen mit Hilfe eines Multiplexers zu einem 32x4 ROM umwandeln?