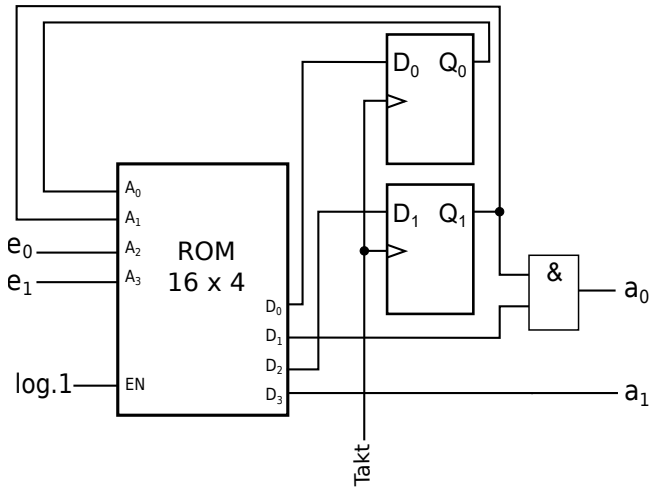
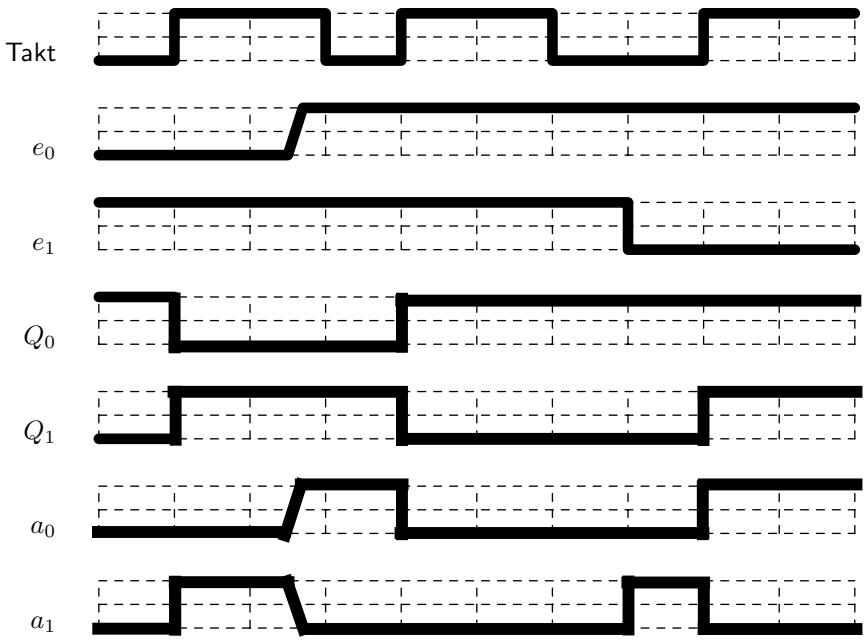
Aufgabe 1: Timing Diagramm

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	1	1
0	1	1	0	1	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	1	1
1	0	0	1	0	1	1	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	1	1
1	1	0	0	0	1	1	0
1	1	0	1	0	1	1	0
1	1	1	0	0	0	1	1
1	1	1	1	1	1	0	0

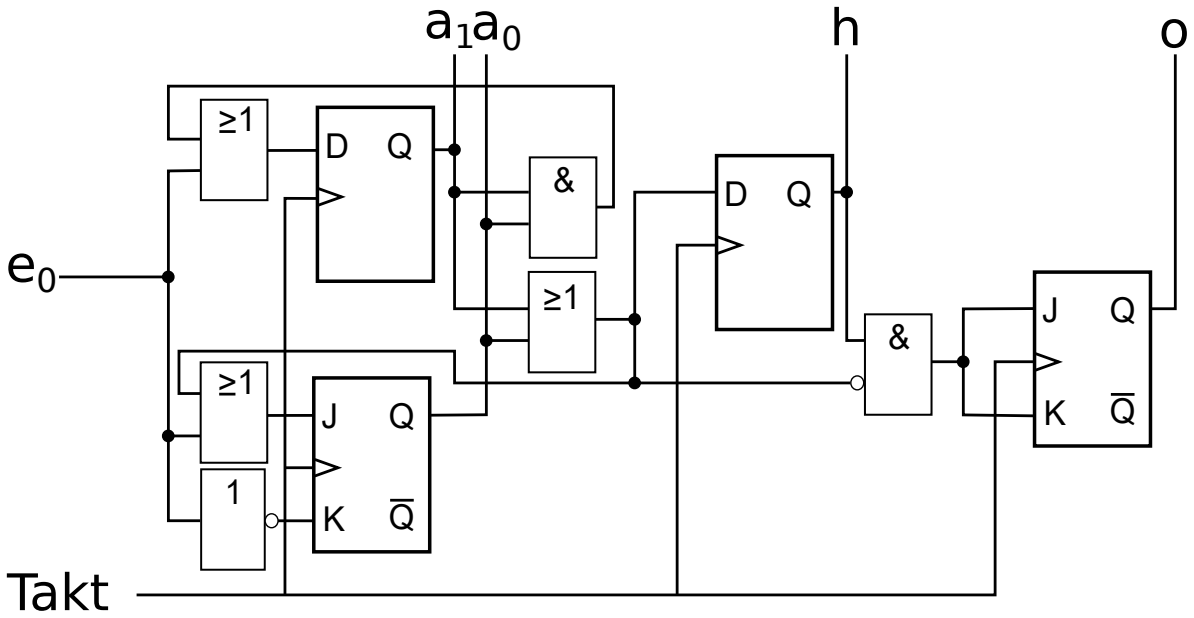


Zeichnen Sie den Verlauf der Signale Q_0, Q_1, a_0 und a_1. Die Durchlaufzeiten des ROM-Bausteins sowie des AND-Gatters sollen dabei vernachlässigt werden.

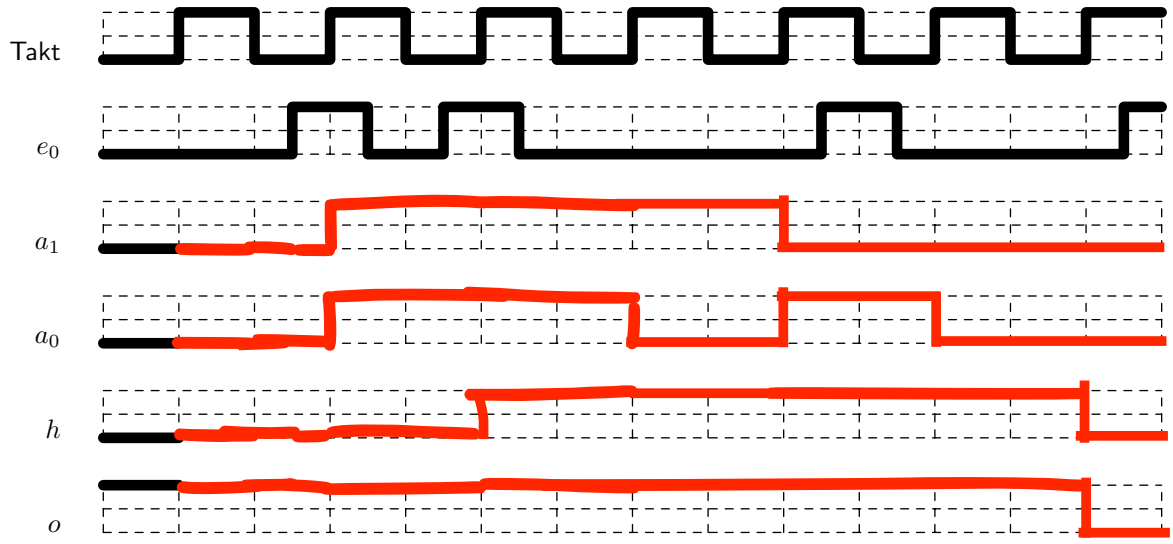


Aufgabe 2: Timing-Diagramm

Es ist folgende Schaltung gegeben:

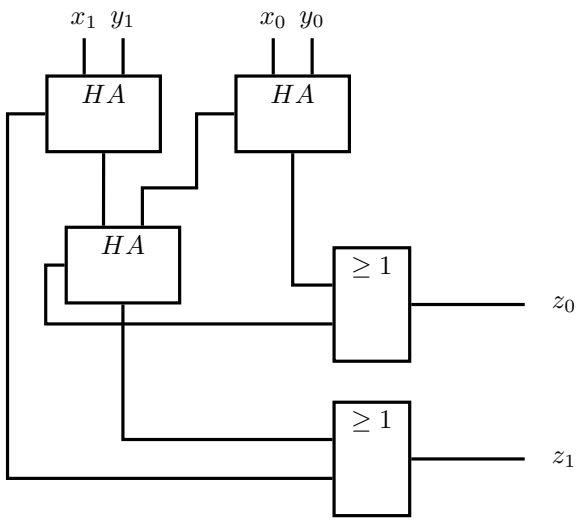


Überlegen Sie sich die Funktionsweise der Schaltung, sodass Sie diese in der Übung erklären können und vervollständigen Sie das nachfolgende Timing-Diagramm. Achten Sie dabei auf die Flankentriggerung.



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}.

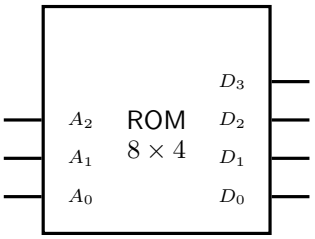


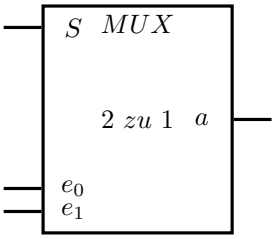
a) Analysieren Sie die oben gegebene Schaltung und vervollständigen Sie den Vordruck der Wahrheitstabelle.

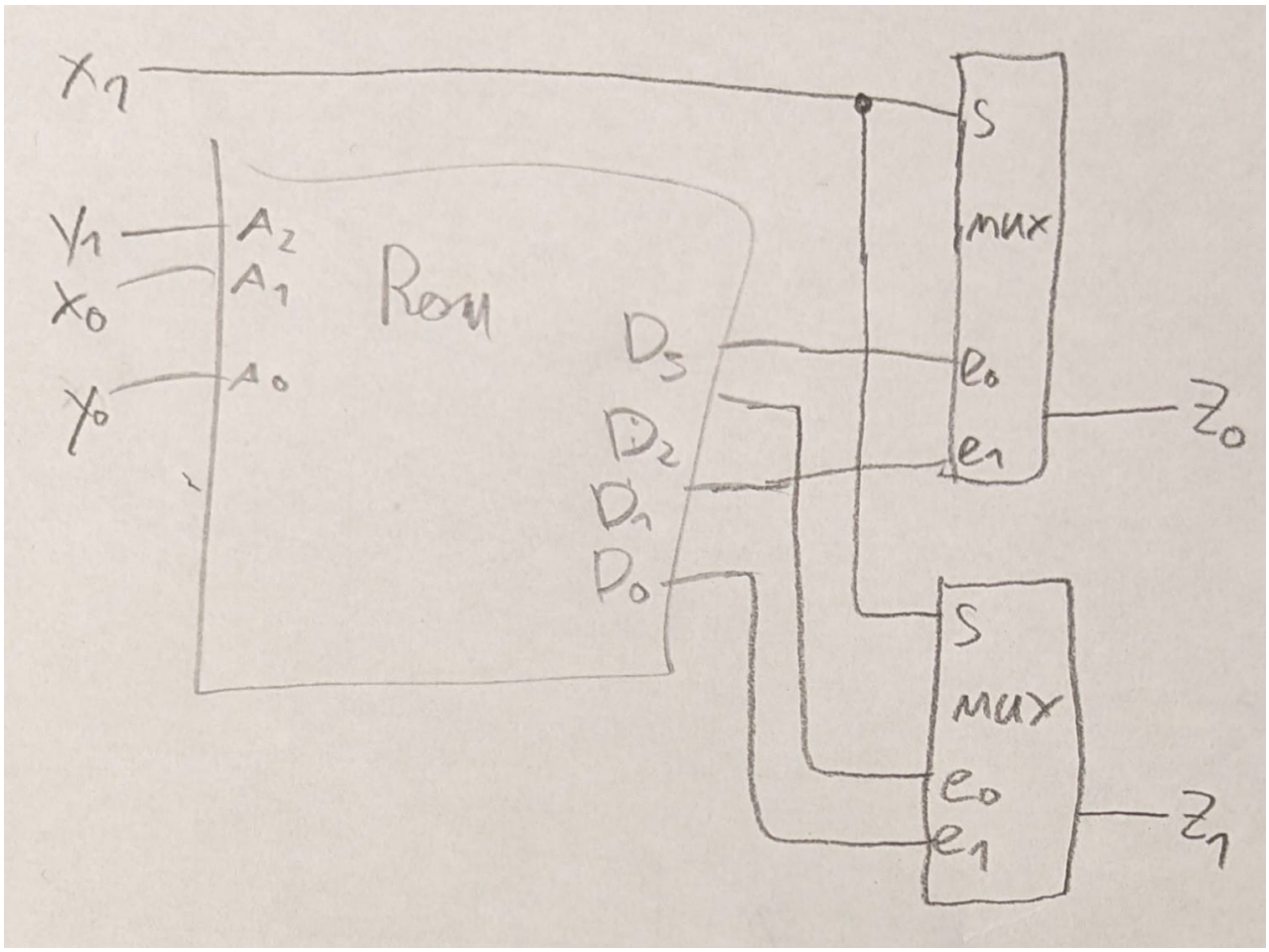
x_1	y_1	x_0	y_0	z_0	z_1
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	0	1

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	1
0	0	1	1	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	0	0	1





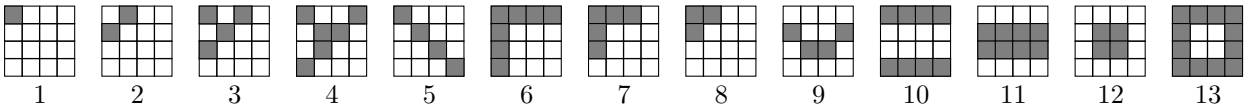


Aufgabe 4: Entwurf & Realisierung einer Boole'schen Funktion via PLA

Entwerfen Sie einen Teil eines Decoders, welcher die dargstellte 4x4 LED-Matrix ansteuert. Die LED-Matrix besitzt 16 Eingänge (s_0 bis s_{15}), welche jeweils die angegebene LED kontrollieren (wobei eine LED leuchtet wenn am entsprechenden Eingang logisch 1 anliegt).

s_0	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	s_{15}

Der Decoder besitzt die Eingänge e_3, e_2, e_1, e_0, welche ein auf der LED-Matrix darzustellendes Muster codieren (e_3 ist das MSB). Die verfügbaren Muster sind unten angegeben. Grau eingefärbte Felder symbolisieren leuchtende LEDs, weiße Felder inaktive LEDs. Die Binärcodierung der unter dem Muster dargestellten Zahl entspricht der Codierung des Musters. Entspricht die in e_3, e_2, e_1 und e_0 dargestellte Zahl keinem Muster, soll kein LED leuchten.



Realisieren Sie eine **minimale** Schaltung für die LEDs an den Positionen s_0, s_4, s_{10} (d.h. die drei Ausgänge Ihrer Schaltung sollen die Werte für diesen drei Eingängen der LED-Matrix entsprechen).

a) Vervollständigen Sie die nachfolgende Wahrheitstabelle. Leiten Sie eine boolesche Funktion in der notwendigen Minimalform von der Wahrheitstabelle ab. Sie benötigen diese für Unteraufgabe b). Sie können dafür beliebige Werkzeuge verwenden. Wir empfehlen den Karnaugh-Veitch Map Rechner[1] entwickelt an der Uni Marburg. Sie müssen die Funktionsweise des verwendeten Werkzeuges nicht erklären. Nur die Eigenschaften des Ergebnisses (Normalform).

x3 e_3	x2 e_2	x1 e_1	x0 e_0	s_0	s_4	s_{10}
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	0	1	1	1	0	0
0	1	0	0	1	0	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	1	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	0	1	1
1	1	0	0	0	0	1
1	1	0	1	1	1	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0

Ausgang „s_0“:

$$(\bar{x}3x0) \vee (\bar{x}3x2) \vee (x3\bar{x}2\bar{x}0) \vee (x2\bar{x}1x0)$$

Ausgang „s_4“:

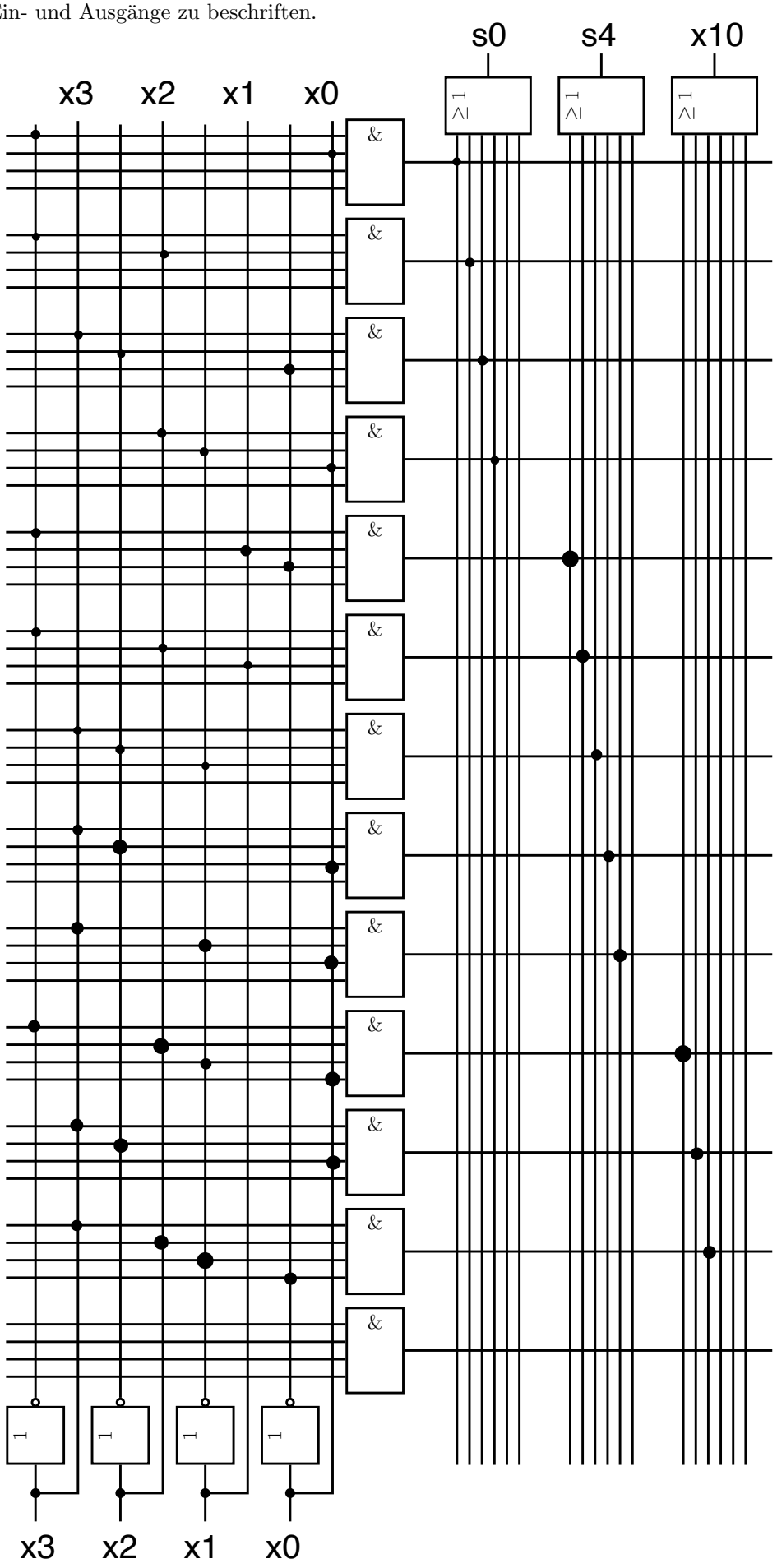
$$(\bar{x}3x1\bar{x}0) \vee (\bar{x}3x2x1) \vee (x3\bar{x}2\bar{x}1) \vee (x3\bar{x}2x0) \vee (x3\bar{x}1x0)$$

Ausgang „s_{10}“:

$$(\bar{x}3x2\bar{x}1x0) \vee (x3\bar{x}2x0) \vee (x3x2\bar{x}1\bar{x}0)$$

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

b) Realisieren Sie den Decoder für die drei Werte s_0, s_4 und s_{10} im unten dargestellten PLA. Vergessen Sie nicht Ein- und Ausgänge zu beschriften.

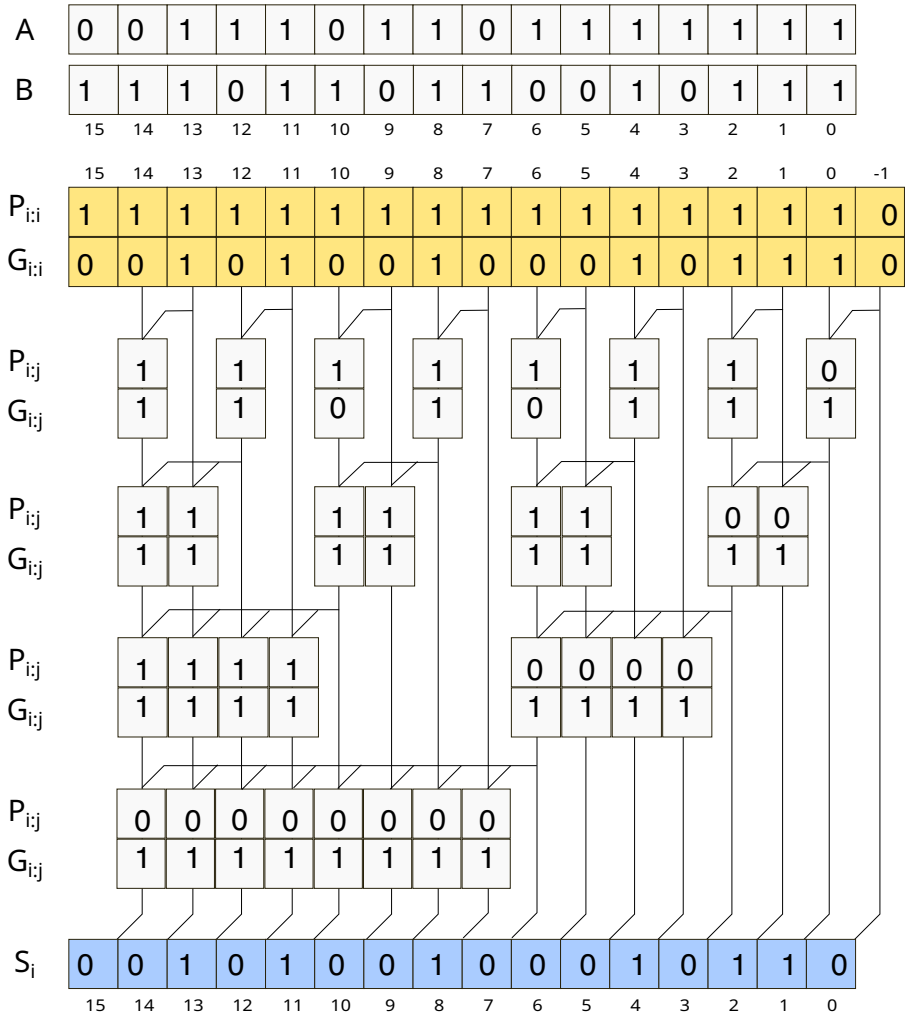


Aufgabe 5: Prefix-Addierer

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

Es wird für jede Ziffer gleichzeitig ausgerechnet, ob an dieser Stelle ein Übertrag generiert wird (G) und ob ein eingehender Übertrag weiter gegeben wird (P). Anschließend werden immer 2 Blöcke zusammengefasst. Dadurch kann die Zahl in logarithmischer Zeit berechnet werden.

b) Gegeben ist die folgende Skizze eines Prefix-Addierers. Berechnen Sie $(15231)_{10} + (-4713)_{10}$ 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: $t_{ripple} = Nt_{FA}$

Beim Ripple Carry Adder werden einfach mehrere Full Adder hintereinander geschaltet. Das nächste bit muss immer auf das vorherige warten, weil der Carry noch nicht bekannt ist. daher N*t_fa

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

Schritt 1: t_PG
für jedes Bit wird P und G ausgerechnet, das passiert parallel
Schritt 2: t_PG_Block
die P und G werden blockweise zusammengefasst, das passiert auch parallel
Schritt 3: t_and&or
Der Carry wird durch die Blöcke durchgereicht. Das passiert hintereinander daher ist die Laufzeit hierfür die Anzahl der Blöcke (N/k), minus 1 weil uns das letzte Carry out nicht interessiert
Schritt 4: k*t_FA
Jetzt können die Blöcke das Ergebnis parallel berechnen. Innerhalb der Blöcke werden wie beim Ripple Carry die bits nacheinander addiert.

- Prefix-Addierer: $t_{PA} = t_{PG} + log_2 N(t_{PG_{Prefix}}) + t_{xor}$
  Erklären Sie auch wieso der letzte Summand nicht $2t_{xor}$ ist.

Es werden zu Beginn parallel alle P und G ausgerechnet.
Dann werden immer zwei Blöcke zusammengefasst. (In der ersten Ebene zwei Bits) In jeder Ebene können die P und G auch parallel berechnet werden. Durch diesen Aufbau der Schaltung werden die anzahl der Blöcke in jeder Ebene halbiert. So kommt eine Laufzeit von log_2 N zustande.
Am Ende müssen noch die einzelnen bits addiert werden. Das kann auch parallel passieren, da wir in den Schritten davor ja schon den Carry berechnet haben. Die Zeit von xor wird nur ein Mal gezählt, weil A xor B schon gerechnet werden kann bevor alles andere fertig ist. Am Ende müssen wir nur noch (A xor B) xor G rechnen.

b) Nehmen Sie an, dass ein Volladdierer eine Latenz von $300ps$ hat. Weiters, nehmen Sie an, dass Gatter mit 2 Eingängen eine Latenz von $200ps$ 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.

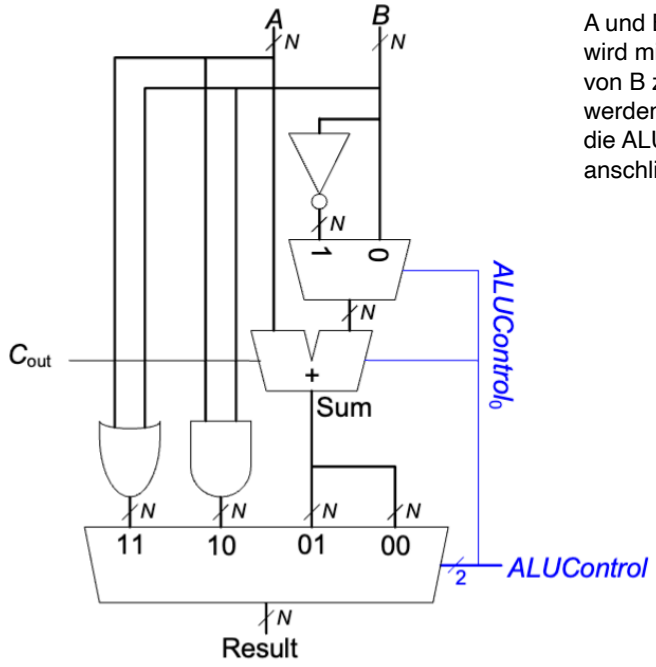
T_ripple = N * 300ps
T_CLA = 200ps + 6 * 200 + (N/4 -1) * 2 * 200ps + 4 * 300ps

N = 4
T_ripple = 1200
T_CLA = 2600
N = 8
T_ripple = 2400
T_CLA = 3000
N = 12
T_ripple = 3600
T_CLA = 3400
ab 12 bits lohnt sich der CLA

Aufgabe 7: Funktionen einer ALU

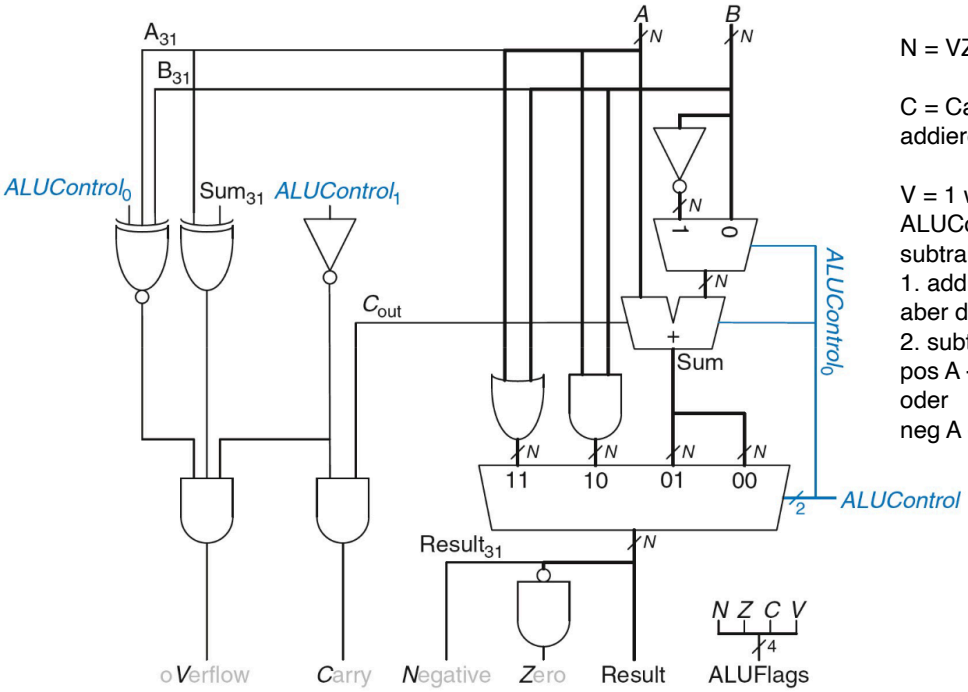
Diese Aufgabe bezieht sich auf die ALU aus der Vorlesung (siehe `03_building_blocks.pdf`, Folie 34 ff.). Seien Sie sich bewusst, dass Sie die Lösung anhand Ihrer Abgabe erklären müssen. Machen Sie daher Skizzen zum Aufbau der ALU, falls das dazu notwendig ist.

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.



A und B stehen im Zweierkomplement. Um zu subtrahieren, wird mit dem negativen von B addiert. Um das Vorzeichen von B zu ändern müssen alle bits geflippt und 1 hinzugefügt werden. Das 1 hinzufügen können wir direkt als Carry in an die ALU übergeben indem wir ALUControl_0 an C_in anschließen.

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



Z = Alle Bits des Erbebnis sind 0

N = VZ Bit ist 1

C = Carry aus der ALU und wir subtrahieren oder addieren

V = 1 wenn
ALUControl_1 = 0 also wir addieren oder subtrahieren und …
1. addieren: A und B sind beide positiv/negativ, aber das Ergebnis ist negativ/positiv
2. subtrahieren:
pos A - neg B und Ergebnis negativ
oder
neg A - pos B und Ergebnis positiv

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**.

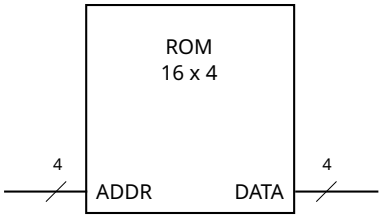


Abbildung 1: 16 x 4 ROM Baustein

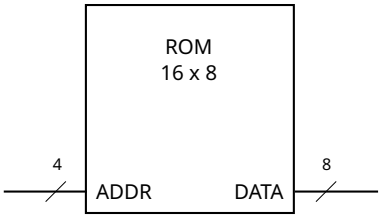


Abbildung 2: 16 x 8 ROM Baustein

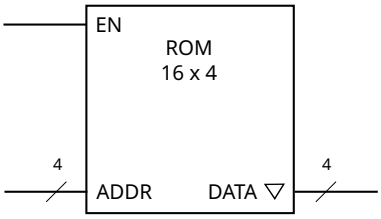
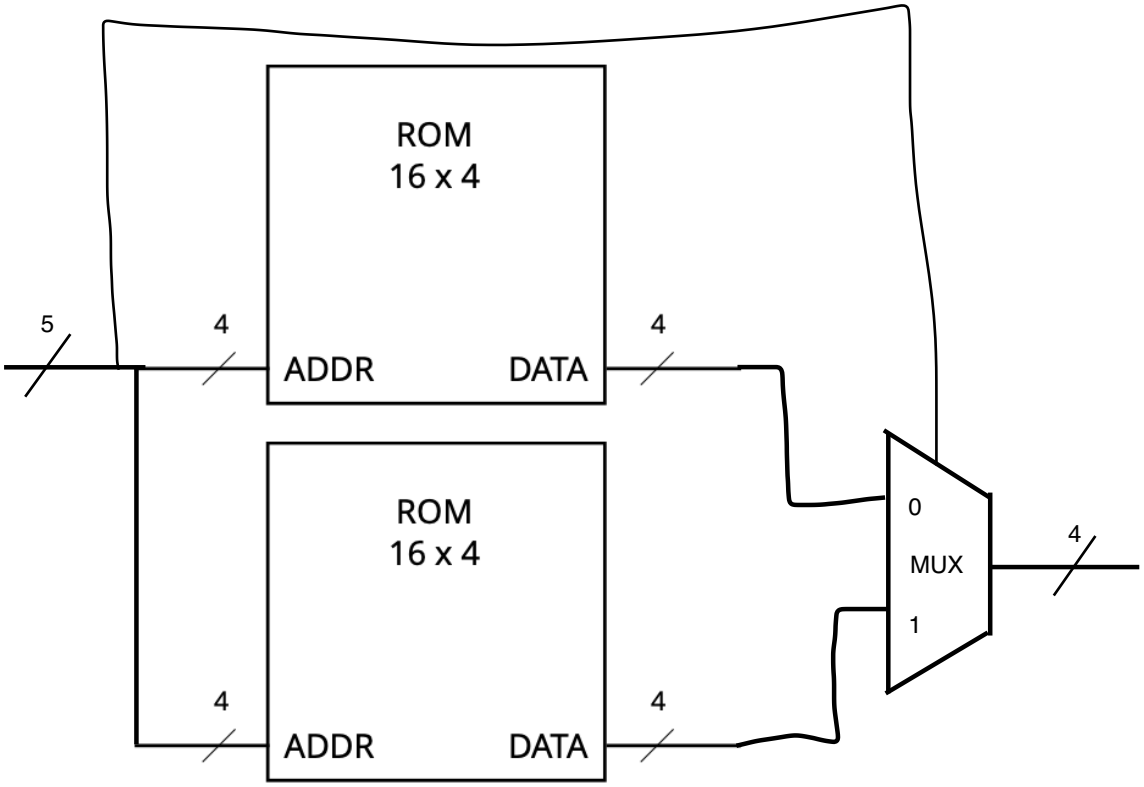
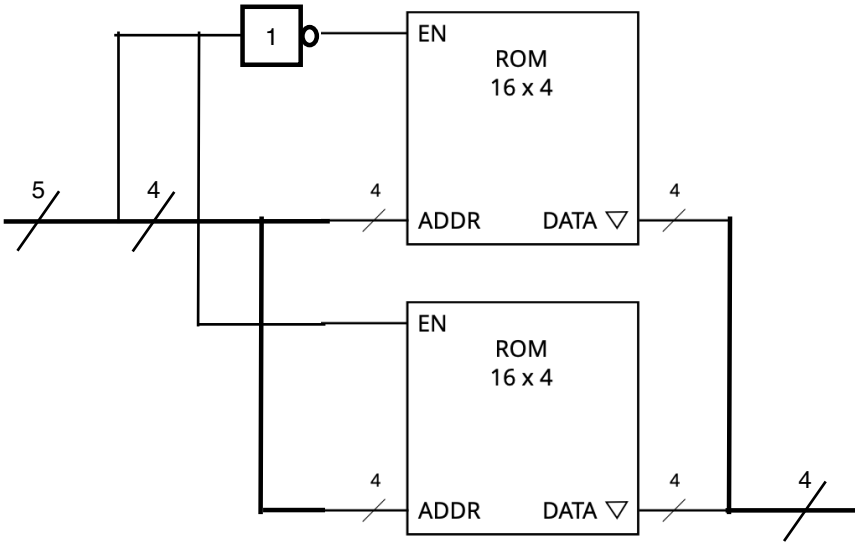


Abbildung 3: 16 x 4 ROM Baustein mit Enable-Eingang

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?