

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 31. Oktober 2013

Technische Informatik

Übungsblatt 1

Aufgabe 1 (3 + 3 + 3 Punkte)

Es sei $\mathbb{B} := \{0, 1\}$. Auf \mathbb{B} seien wie in der Vorlesung vorgestellt die booleschen Operatoren \vee, \wedge und \neg definiert. $+, \cdot$ und $-$ bezeichnen hingegen die üblichen Operatoren aus dem Ring $(\mathbb{Z}, +, \cdot)$:

$$\begin{aligned} x \vee y &:= x + y - x \cdot y \\ x \wedge y &:= x \cdot y \\ \neg x &:= 1 - x \end{aligned}$$

Beweisen Sie formal durch Beweis der folgenden in der Vorlesung vorgestellten Axiome, dass (B, \wedge, \vee, \neg) die folgenden Axiome einer Booleschen Algebra erfüllt

a) Kommutativität	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
b) Assoziativität	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
c) Distributivität	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

Benutzen Sie dabei ausschließlich die obigen Operatordefinitionen und Rechenregeln aus $(\mathbb{Z}, +, \cdot)$, gehen Sie formal vor und nicht über Tabellen wie in der Vorlesung.

Aufgabe 2 (3 Punkte)

In der Vorlesung wurde die Boolesche Algebra $(\{0, 1\}, \wedge, \vee, \neg)$ vorgestellt. Sie haben gesehen, dass für jede Boolesche Algebra folgende Axiome gelten:

(i) Kommutativität	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
(ii) Assoziativität	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
(iii) Absorption	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$
(iv) Distributivität	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
(v) Komplementregel	$x \vee (y \wedge \neg y) = x$	$x \wedge (y \vee \neg y) = x$

Sie haben ebenso einige Folgerungen aus diesen Axiomen kennen gelernt (siehe Kap. 1.1, Folie 17). Beweisen Sie durch ausschließliche Verwendung der Axiome, dass das doppelte Komplement gültig ist:

$$\neg\neg x = x \quad \text{für } x \in \{0, 1\}$$

Geben Sie in jedem Schritt an, welche Regel Sie benutzt haben.

Hinweise:

- Die L^AT_EX-Kommandos für die logischen Operatoren lauten `\land` für \wedge , `\lor` für \vee und `\neg` für \neg .
- Nutzen Sie wirklich ausschließlich die angegebenen Axiome. Ein „Beweis“ durch Funktionstabellen ist nicht zulässig.
- x, y und z sind beliebige Elemente aus der Booleschen Algebra, insbesondere kann auch $x = y$ gelten.

Aufgabe 3 (4 Punkte)

In der Vorlesung haben Sie gerichtete, azyklische Graphen kennengelernt und gehört, dass die Tiefe von G als die Länge des längsten Pfades in G definiert ist (s. Kap. 1.1, Folie 20). Beweisen Sie nun folgenden Satz:

„Der längste Pfad in einem gerichteten, azyklischen Graphen G geht immer von einer Wurzel zu einem Blatt.“

Tipp: Versuchen Sie es mit einem Widerspruchsbeweis.

Aufgabe 4 (4 Punkte)

Zeigen Sie durch vollständige Induktion:

Für alle $n = 2^m$ mit $m \in N$ gibt es einen binären Baum mit 2^m Blättern und Graph-Tiefe $m = \log n$.

Aufgabe 5 (4 Punkte)

Beweisen Sie formal, dass nach Ausführung des folgenden Programms für die Speicherzelle S(2) gilt:

$$S(2) = x \cdot (y + 1)$$

unter den Bedingungen, dass zu Beginn gilt: $S(0) = x; S(1) = y, y > 0$

Zeigen Sie dazu zunächst durch Induktion über n die folgende Aussage:

Falls Zeile 8 im Programmablauf überhaupt n Mal durchlaufen wird, dann gilt nach Durchlauf Nr. n von Zeile 8:

$$ACC = S(1) = y - n \text{ und } S(2) = x \cdot n.$$

Benutzen Sie nun diese Aussage, um die gesamte Anzahl der Durchläufe der Zeile 8 in Abhängigkeit von y zu bestimmen und zeigen Sie damit die Behauptung.

```
1 LOADI 0    ; ACC := 0
2 STORE 2    ; S(2) := 0
3 LOAD 2     ; ACC := S(2)
4 ADD 0      ; ACC := ACC + S(0)
5 STORE 2    ; S(2) := ACC
6 LOAD 1     ; ACC := S(1)
7 SUBI 1     ; ACC := ACC - 1
8 STORE 1    ; S(1) := ACC
9 JUMP> -6  ; PC := PC - 6 if ACC ≥ 0
10 END
```

Abgabe: 8. November 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 31. Oktober 2013

Technische Informatik

Musterlösung zu Übungsblatt 1

Aufgabe 1 (3 + 3 + 3 Punkte)

Es sei $\mathbb{B} := \{0, 1\}$. Auf \mathbb{B} seien wie in der Vorlesung vorgestellt die booleschen Operatoren \vee, \wedge und \neg definiert. $+, \cdot$ und $-$ bezeichnen hingegen die üblichen Operatoren aus dem Ring $(\mathbb{Z}, +, \cdot)$:

$$\begin{aligned} x \vee y &:= x + y - x \cdot y \\ x \wedge y &:= x \cdot y \\ \neg x &:= 1 - x \end{aligned}$$

Beweisen Sie formal durch Beweis der folgenden in der Vorlesung vorgestellten Axiome, dass (B, \wedge, \vee, \neg) die folgenden Axiome einer Booleschen Algebra erfüllt

a) Kommutativität	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
b) Assoziativität	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
c) Distributivität	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

Benutzen Sie dabei ausschließlich die obigen Operatordefinitionen und Rechenregeln aus $(\mathbb{Z}, +, \cdot)$, gehen Sie formal vor und nicht über Tabellen wie in der Vorlesung.

Lösung:

Desöfteren verwendet: $\forall z \in \mathbb{B}: z^2 = z$

a) Kommutativität

$$\begin{aligned} x \vee y &= x + y - x \cdot y \\ &= y + x - y \cdot x \\ &= y \vee x \\ x \wedge y &= x \cdot y \\ &= y \cdot x \\ &= y \wedge x \end{aligned}$$

b) Assoziativitat

$$\begin{aligned}
 x \vee (y \vee z) &= x + (y \vee z) - x \cdot (y \vee z) \\
 &= x + y + z - yz - x(y + z - yz) \\
 &= x + y + z - yz - xy - xz + xyz \\
 &= x + y - xy + z - (xz + yz - xyz) \\
 &= (x \vee y) + z - z(x + y - xy) \\
 &= (x \vee y) + z - z(x \vee y) \\
 &= (x \vee y) \vee z
 \end{aligned}$$

$$\begin{aligned}
 x \wedge (y \wedge z) &= x \cdot (y \cdot z) \\
 &= (x \cdot y) \cdot z \\
 &= (x \wedge y) \wedge z
 \end{aligned}$$

c) Distributivitat

$$\begin{aligned}
 (x \vee y) \wedge (x \vee z) &= (x + y - xy) \cdot (x + z - xz) \\
 &= x(x + z - xz) + y(x + z - xz) - xy(x + z - xz) \\
 &= x^2 + xz - x^2z + xy + yz - xyz - x^2y - xyz + x^2yz \\
 &= x + xz - xz + xy + yz - xyz - xy - xyz + xyz \\
 &= x + yz - xyz \\
 &= x + (y \cdot z) - x \cdot (y \cdot z) \\
 &= x \vee (y \wedge z)
 \end{aligned}$$

$$\begin{aligned}
 (x \wedge y) \vee (x \wedge z) &= xy + xz - (xy \cdot xz) \\
 &= x(y + z - yz) \\
 &= x \wedge (y \vee z)
 \end{aligned}$$

Aufgabe 2 (3 Punkte)

In der Vorlesung wurde die Boolesche Algebra ($\{0, 1\}, \wedge, \vee, \neg$) vorgestellt. Sie haben gesehen, dass fur jede Boolesche Algebra folgende Axiome gelten:

- | | | |
|-----------------------|--|--|
| (i) Kommutativitat | $x \vee y = y \vee x$ | $x \wedge y = y \wedge x$ |
| (ii) Assoziativitat | $x \vee (y \vee z) = (x \vee y) \vee z$ | $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ |
| (iii) Absorption | $x \vee (x \wedge y) = x$ | $x \wedge (x \vee y) = x$ |
| (iv) Distributivitat | $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ | $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ |
| (v) Komplementregel | $x \vee (y \wedge \neg y) = x$ | $x \wedge (y \vee \neg y) = x$ |

Sie haben ebenso einige Folgerungen aus diesen Axiomen kennen gelernt (siehe Kap. 1.1, Folie 17). Beweisen Sie durch ausschlieliche Verwendung der Axiome, dass das doppelte Komplement gultig ist:

$$\neg\neg x = x \quad \text{fur } x \in \{0, 1\}$$

Geben Sie in jedem Schritt an, welche Regel Sie benutzt haben.

Hinweise:

- Die L^AT_EX-Kommandos für die logischen Operatoren lauten \land für \wedge , \lor für \vee und \neg für \neg .
- Nutzen Sie wirklich ausschließlich die angegebenen Axiome. Ein „Beweis“ durch Funktionstabellen ist nicht zulässig.
- x, y und z sind beliebige Elemente aus der Booleschen Algebra, insbesondere kann auch $x = y$ gelten.

Lösung:

(i) Kommutativität	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
(ii) Assoziativität	$x \vee (y \vee z) = (x \vee b) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
(iii) Absorption	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = a$
(iv) Distributivität	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	$x \wedge (y \vee c) = (x \wedge y) \vee (x \wedge z)$
(v) Komplementregel	$x \vee (y \wedge \neg y) = x$	$x \wedge (y \vee \neg y) = x$

Punkte: Beweis durch Funktionstabelle [0]; Vergessen die Kommutativität anzugeben: einmalig [-0,5].

$$\begin{aligned}
 \neg\neg x &\stackrel{(v)}{=} \neg\neg x \wedge (x \vee \neg x) \\
 &\stackrel{(iv)}{=} (\neg\neg x \wedge x) \vee (\neg\neg x \wedge \neg x) \\
 &\stackrel{(v)}{=} (\neg\neg x \wedge x) \\
 &\stackrel{(v)}{=} (\neg\neg x \wedge x) \vee (\neg x \wedge x) \\
 &\stackrel{(iv)+(i)}{=} x \wedge (\neg\neg x \vee \neg x) \\
 &\stackrel{(v)}{=} x
 \end{aligned}$$

Aufgabe 3 (4 Punkte)

In der Vorlesung haben Sie gerichtete, azyklische Graphen kennengelernt und gehört, dass die Tiefe von G als die Länge des längsten Pfades in G definiert ist (s. Kap. 1.1, Folie 20). Beweisen Sie nun folgenden Satz:

„Der längste Pfad in einem gerichteten, azyklischen Graphen G geht immer von einer Wurzel zu einem Blatt.“

Tipp: Versuchen Sie es mit einem Widerspruchsbeweis.

Lösung:

Korrekturhinweis: Es handelt sich hier um einen eher argumentativen Beweis. Es ist deswegen weniger dramatisch, wenn die Formalia nicht stimmen, als wenn es Fehler in der Argumentation gibt.

Annahme: Es gibt einen längsten Pfad π in einem gerichteten, azyklischen Graphen G , der nicht von einer Wurzel zu einem Blatt geht.

Fall 1: Der Startzustand s_I von π ist ein innerer Knoten. Da es sich dabei nicht um eine Wurzel handelt, muss gelten: $\text{indeg}(s_I) > 0$.

$\Rightarrow s_I$ hat mindestens einen Vorgänger.

$\Rightarrow \pi$ lässt sich verlängern, indem man einen der Vorgänger von s_I als Startzustand wählt.

\Rightarrow *Widerspruch*, da π längster Pfad.

Fall 2: Der Endzustand s_N ist ein innerer Knoten. Da es sich dabei nicht um ein Blatt handelt, muss gelten: $\text{outdeg}(s_N) > 0$.

$\Rightarrow s_N$ hat mindestens einen Nachfolger.

$\Rightarrow \pi$ lässt sich verlängern, indem man einen der Nachfolger von s_N hinzufügt.

\Rightarrow *Widerspruch*, da π längster Pfad.

Fall 1 und Fall 2 führen zum Widerspruch \Rightarrow Die Annahme ist falsch.

\Rightarrow Der längste Pfad in einem gerichteten, azyklischen Graphen führt immer von einer Wurzel zu einem Blatt.

q.e.d

Aufgabe 4 (4 Punkte)

Zeigen Sie durch vollständige Induktion:

Für alle $n = 2^m$ mit $m \in N$ gibt es einen binären Baum mit 2^m Blättern und Graph-Tiefe $m = \log n$.

Lösung:

Induktion über m .

$m = 0$ (Induktions-Anfang):

Der Baum hat $2^0 = 1$ Blatt ($2^{0+1} - 1 = 1$ Knoten), und Tiefe 0 weil man auf keiner Kante vom Wurzel durchlaufen muss

$m > 0$ (Induktions-Schritt):

Induktionsvoraussetzung: $m \rightarrow m + 1$

Induktionsbehauptung: für m hat der Baum 2^m Blätter und Tiefe m

Induktionsbeweis:

Wir konstruieren einen binären Baum mit 2 binären Bäumen der Tiefe m , die jeweilige Wurzeln werden mit einem neuen Knoten verbunden. Der neue Baum hat dann Tiefe $m + 1$ und $2 \cdot 2^m = 2^{(m+1)}$ Blätter.

Aufgabe 5 (4 Punkte)

Beweisen Sie formal, dass nach Ausführung des folgenden Programms für die Speicherzelle S(2) gilt:

$$S(2) = x \cdot (y + 1)$$

unter den Bedingungen, dass zu Beginn gilt: $S(0) = x; S(1) = y, y > 0$

Zeigen Sie dazu zunächst durch Induktion über n die folgende Aussage:

Falls Zeile 8 im Programmablauf überhaupt n Mal durchlaufen wird, dann gilt nach Durchlauf Nr. n von Zeile 8:

$$ACC = S(1) = y - n \text{ und } S(2) = x \cdot n.$$

Benutzen Sie nun diese Aussage, um die gesamte Anzahl der Durchläufe der Zeile 8 in Abhängigkeit von y zu bestimmen und zeigen Sie damit die Behauptung.

```
1 LOADI 0    ; ACC := 0
2 STORE 2    ; S(2) := 0
3 LOAD 2     ; ACC := S(2)
4 ADD 0      ; ACC := ACC + S(0)
5 STORE 2    ; S(2) := ACC
6 LOAD 1     ; ACC := S(1)
7 SUBI 1     ; ACC := ACC - 1
8 STORE 1    ; S(1) := ACC
9 JUMP≥ -6 ; PC := PC - 6 if ACC ≥ 0
10 END
```

Lösung:

Induktionsbeginn (Basisfall) für $n = 1$ ([1,5] Punkte).
Man muss das Programm von Zeile 1 bis 8 ausführen:

Man sieht dann: Die Zeile 8 wird ein Mal durchlaufen und es gilt $S(1) = y - 1$ und $S(2) = x$.

Induktionsvoraussetzung (IV): Für ein y gilt $S(2) = x \cdot (y + 1)$.

Induktionsschritt: $y \rightarrow y + 1$ ([3] Punkte).

Zu zeigen: Falls Zeile 8 im Programmablauf überhaupt $n + 1$ Mal durchlaufen wird, dann gilt nach Durchlauf Nr. $n + 1$ von Zeile 8: $ACC = S(1) = y - (n + 1)$ und $S(2) = x \cdot (n + 1)$.

2 Fälle:

- Zeile 8 wird nicht $n + 1$ Mal durchlaufen. Dann ist nichts zu zeigen.
- Zeile 8 wird $n + 1$ Mal durchlaufen. Dann wird sie natürlich auch n Mal durchlaufen und nach I.V. gilt nach Durchlauf Nr. n von Zeile 8: $ACC = S(1) = y - n$ und $S(2) = x \cdot n$. Da die Zeile 8 nach Fallannahme $n + 1$ Mal durchlaufen wird, muss nach dem n . Durchlauf von Zeile 8 der Sprung von Zeile 9 nach Zeile 3 genommen werden. Durch weitere Auswertung der Zeilen 3 – 8 erhält man $ACC = S(1) = y - (n + 1)$ und $S(2) = x \cdot (n + 1)$.

Damit ist der Induktionsbeweis zu Ende und es ist noch zu zeigen: Nach Ausführung des Programms gilt: $S(2) = x \cdot (y + 1)$

```

1 LOADI 0 ; ACC := 0
2 STORE 2 ; S(2) := 0
3 LOAD 2 ; ACC := S(2)
4 ADD 0 ; ACC := ACC + S(0) = 0 + x = x
5 STORE 2 ; S(2) := ACC = x
6 LOAD 1 ; ACC := S(1) = 1
7 SUBI 1 ; ACC := 0
8 STORE 1 ; S(1) := ACC = 0
9 JUMP> -6 ; PC := PC - 6 if ACC  $\geq 0$  // Sprungbedingung erfüllt
3 LOAD 2 ; ACC := S(2) = x
4 ADD 0 ; ACC := ACC + S(0) = x+x = 2·x
5 STORE 2 ; S(2) := ACC = 2·x
6 LOAD 1 ; ACC := 0
7 SUBI 1 ; ACC := ACC - 1 = -1
8 STORE 1 ; S(1) := ACC = -1
9 JUMP> -6 ; PC := PC - 6 if ACC  $\geq 0$  // Sprungbedingung nicht erfüllt
10 END

```

Anhand der bewiesenen Aussage sieht man leicht ein, dass für die Durchläufe 1 bis y $ACC \geq 0$ gilt, und der Sprung von Zeile 9 nach Zeile 3 jeweils genommen wird (die Aussage „Für die Durchläufe $i = 1, \dots, y$ gilt $ACC \geq 0$ und der Sprung von Zeile 9 nach Zeile 3 wird jeweils genommen“ könnte man streng genommen nochmals „induktiv“ beweisen. Das ist aber keine so richtige „Standardinduktion“, da man bei Basisfall darauf achten muss, dass man nur was zeigen muss, wenn $1 \leq y$, und beim Induktionsschritt nur, wenn $i+1 \leq y$).

Die Zeile 8 wird also auch $y+1$ Mal durchlaufen und nach dem Durchlauf Nr. $y+1$ gilt:
 $ACC = S(1) = -1$, $S(2) = x \cdot (y+1)$.

Dann wird in Zeile 9 der Sprung aber nicht genommen und das Programm ist zu Ende.

```

3 LOAD 2 ; ACC := S(2) IV = x·(y+1)
4 ADD 0 ; ACC := ACC + S(0) = x·(y+1)+x = x·(y+2)
5 STORE 2 ; S(2) := ACC = x·(y+2)
6 LOAD 1 ; ACC := * (y+1)-(y+1) = 0
7 SUBI 1 ; ACC := ACC - 1 = -1
8 STORE 1 ; S(1) := ACC = -1
9 JUMP> -6 ; PC := PC - 6 if ACC  $\geq 0$  // Sprungbedingung nicht erfüllt
10 END

```

Für Formfehler (Induktionsbeginn, -voraussetzung, -schritt nicht angegeben oder falsch benutzt) [-0.5].

Abgabe: 8. November 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 08. November 2013

Technische Informatik Übungsblatt 2

Aufgabe 1 (3 + 1 + 1 Punkte)

Gegeben sei ein Alphabet $A = \{a, b, e, f, g, i, m, r, s, u\}$ und folgende Häufigkeitsverteilung:

Zeichen	a	b	e	f	g	i	m	r	s	u
Häufigkeit in %	30	21	14	9	8	6	5	4	2	1

- a) Erzeugen Sie für die Häufigkeitsverteilung einen Huffman-Code. Gehen Sie analog zur Vorlesung vor.
- b) Kodieren Sie den Text `freiburg im breisgau` mit dem erzeugten Huffman-Code. Ignorieren Sie die Leerzeichen.
- c) Gibt es Präfixcodes, die das gegebene Textbeispiel kürzer kodieren? Begründen Sie kurz Ihre Antwort.

Aufgabe 2 (2 + 2 + 2 + 1 Punkte)

Sei ein Alphabet $A = a_1, \dots, a_m$ mit $m \geq 3$ gegeben. Sei weiterhin eine Häufigkeitsverteilung p gegeben, die jedem Zeichen $a_i \in A$ eine Häufigkeit zuordnet mit $\sum_{i=1}^m p(a_i) = 1$. Der zugehörige Huffman-Code sei mit c bezeichnet.

- a) Zeigen Sie: Falls es ein $i \in 1, \dots, m$ gibt mit $p(a_i) > 0.5$, dann gilt $|c(a_i)| = 1$ (d.h. das Codewort, das a_i zugeordnet wird, hat die Länge 1).
- b) Zeigen Sie: Falls es ein $i \in 1, \dots, m$ gibt mit $|c(a_i)| = 1$, dann muss $p(a_i) \geq 1/3$ gelten.
(Hinweis: Beweis durch Widerspruch.)
- c) In der Vorlesung wurde bei der Huffman-Codierung angegeben: „Die linken Kanten werden mit 0 markiert und die rechten Kanten mit 1.“ Ändere nun diese Forderung in „An jedem Knoten wird eine der ausgehenden Kanten mit 0 und eine mit 1 markiert.“ Funktioniert die Dekodierung bei dieser geänderten Huffman-Codierung und wenn ja unter welchen Voraussetzungen?

- d) Nehmen Sie nun eine Häufigkeitsverteilung an, die für jedes Zeichen des Alphabets den gleichen Wert ergibt, d.h. $P(a_i) = P(a_j)$ für $1 \leq i < j \leq m$. Nehmen Sie an, Sie würden einen Huffman-Code erzeugen. Wie groß wäre nun die mittlere Codelänge? Begründen Sie Ihre Antwort.

Aufgabe 3 (1 + 4 Punkte)

- a) Sei $[1001]_2$ eine Zahl in Zweier-Komplement-Darstellung *ohne Nachkommastellen*. Um welche ganze Zahl (mit der Basis 10) handelt es sich?
- b) Beweisen Sie folgendes Lemma:

Lemma: Sei $[a]_2 = [a_{n-1}a_{n-2}\dots a_0]_2$ eine *ganze* Zahl in Zweier-Komplement-Darstellung mit n Vorkommastellen und *keinen* Nachkommastellen. Dann gilt:

$$[\bar{a}]_2 + 1 = -[a]_2$$

Hierbei sei $[\bar{a}]_2$ die Zahl im Zweier-Komplement, die aus $[a]_2$ durch Invertieren aller Bits hervorgeht.

Abgesehen von der geometrischen Summenformel sollen keine Sätze aus der Vorlesung ohne Beweis benutzt werden.

Aufgabe 4 (3 Punkte)

Geben Sie eine einfache Methode an zum Umwandeln einer natürlichen Zahl aus Binärdarstellung in Hexadezimaldarstellung und umgekehrt ohne den Zwischenschritt der Umwandlung in das Dezimalsystem.

Aufgabe 5 (4 Punkte)

Konvertieren Sie die Zahlen in Betrag & Vorzeichen, Einerkomplement und Zweierkomplement Darstellung.

$$-234_{10} \quad 84_{10} \quad FCB1_{16} \quad -1_{10} \quad -10_{16} \quad -654_8 \quad -FF_{16} \quad 963_{10} \quad -255_{10}$$

Abgabe: 15. November 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 08. November 2013

Technische Informatik Musterlösung zu Übungsblatt 2

Aufgabe 1 (3 + 1 + 1 Punkte)

Gegeben sei ein Alphabet $A = \{a, b, e, f, g, i, m, r, s, u\}$ und folgende Häufigkeitsverteilung:

Zeichen	a	b	e	f	g	i	m	r	s	u
Häufigkeit in %	30	21	14	9	8	6	5	4	2	1

- Erzeugen Sie für die Häufigkeitsverteilung einen Huffman-Code. Gehen Sie analog zur Vorlesung vor.
- Kodieren Sie den Text `freiburg im breisgau` mit dem erzeugten Huffman-Code. Ignorieren Sie die Leerzeichen.
- Gibt es Präfixcodes, die das gegebene Textbeispiel kürzer kodieren? Begründen Sie kurz Ihre Antwort.

Lösung:

- Baum wird in der Tutorenbesprechung gemahlt.

a	00
b	10
e	010
f	110
g	0110
i	1110
m	1111
r	01110
s	011110
u	011111

Für den Baum bekommt man 3 Punkte. Jeder Fehler -0,5 Punkte.

Für den Code bekommt man 1 Punkt. Sollte der Baum falsch sein, der Code aber korrekt gebildet werden, erhält man trotzdem einen Punkt (Folgefehler).

- b)

$\overbrace{f}^{110} \overbrace{r}^{01110} \overbrace{e}^{010} \overbrace{i}^{1110} \overbrace{b}^{10} \overbrace{u}^{011111} \overbrace{r}^{011110} \overbrace{g}^{0110} \overbrace{i}^{1110} \overbrace{m}^{1111} \overbrace{b}^{10} \overbrace{r}^{01110} \overbrace{e}^{010} \overbrace{i}^{1110} \overbrace{s}^{011110} \overbrace{g}^{0110} \overbrace{a}^{00} \overbrace{u}^{011111}$

- c) Ja, da die angenommene Häufigkeitsverteilung nicht der konkreten Verteilung im Beispiel entspricht.

Aufgabe 2 (2 + 2 + 2 + 1 Punkte)

Sei ein Alphabet $A = a_1, \dots, a_m$ mit $m \geq 3$ gegeben. Sei weiterhin eine Häufigkeitsverteilung p gegeben, die jedem Zeichen $a_i \in A$ eine Häufigkeit zuordnet mit $\sum_{i=1}^m p(a_i) = 1$. Der zugehörige Huffman-Code sei mit c bezeichnet.

- Zeigen Sie: Falls es ein $i \in 1, \dots, m$ gibt mit $p(a_i) > 0.5$, dann gilt $|c(a_i)| = 1$ (d.h. das Codewort, das a_i zugeordnet wird, hat die Länge 1).
- Zeigen Sie: Falls es ein $i \in 1, \dots, m$ gibt mit $|c(a_i)| = 1$, dann muss $p(a_i) \geq 1/3$ gelten.
(Hinweis: Beweis durch Widerspruch.)
- In der Vorlesung wurde bei der Huffman-Codierung angegeben: „Die linken Kanten werden mit 0 markiert und die rechten Kanten mit 1.“ Ändere nun diese Forderung in „An jedem Knoten wird eine der ausgehenden Kanten mit 0 und eine mit 1 markiert.“ Funktioniert die Dekodierung bei dieser geänderten Huffman-Codierung und wenn ja unter welchen Voraussetzungen?
- Nehmen Sie nun eine Häufigkeitsverteilung an, die für jedes Zeichen des Alphabets den gleichen Wert ergibt, d.h. $P(a_i) = P(a_j)$ für $1 \leq i < j \leq m$. Nehmen Sie an, Sie würden einen Huffman-Code erzeugen. Wie groß wäre nun die mittlere Codelänge? Begründen Sie Ihre Antwort.

Lösung:

- Gegeben:

$$\sum_{j=1}^m p(a_j) = 1, p(a_i) > 0.5$$

dann folgt:

$$\begin{aligned} \sum_{j=1}^{i-1} p(a_j) + p(a_i) + \sum_{j=i+1}^m p(a_j) &= 1 \\ \sum_{j=1}^{i-1} p(a_j) + \sum_{j=i+1}^m p(a_j) &< 0.5 \end{aligned}$$

Beim Bau der binären Baums sind alle Häufigkeitsteilsummen kleiner als $p(a_i)$, daher wird a_i zu dem Baum addiert, nur wenn alle anderen Knoten bereits zusammen addiert sind, zu dem wird ein neuer Knoten (Wurzel) durch eine 1-Kante addiert, welche 0-Kante führt direkt zu a_i .

- Annahme: $p(a_i) < 1/3$ und $|c(a_i)| = 1$.
Wegen $|c(a_i)| = 1$ ist a_i einer der beiden direkten Vorgänger der Wurzel des Codebaumes. Für den anderen Vorgänger v gilt:
 - Er ist mit einer Häufigkeitssumme $> 2/3$ beschriftet, da die Summe an der Wurzel 1 ist und $p(a_i) < 1/3$
 - Der Vorgänger v ist kein Blatt wegen $m \geq 3$.

Also hat v mindestens einen Vorgänger v_1 mit Häufigkeitssumme größer als $1/3$. Der Algorithmus hätte dann aber zum Zeitpunkt des Einfügens von v nicht v_1 , sondern einen Knoten mit einer kleineren Häufigkeitssumme als Vorgänger von v auswählen müssen. Ein solcher Kandidat wäre z.B. a_i mit $p(a_i) < 1/3$ gewesen. Widerspruch!

- Funktioniert, wenn dem Dekodierer die Abbildung von Zeichen auf Codewörter bekannt ist. Dann kann man den Baum rekonstruieren und mit dem angegebenen Verfahren dekodieren. (Insbesondere ist der resultierende Code auch ein Präfixcode.)
- Aufgrund der identischen Häufigkeiten würde beim Aufbau des Huffman-Codes ein balancierter Binärbaum entstehen. Die Anzahl der Blätter ist m . Daraus folgt, dass jeder Pfad in diesem Baum die Länge i hat wegen $m = 2^i$. Damit hätte jedes Codewort die Länge i und es ergibt sich die mittlere Codelänge i .

Aufgabe 3 (1 + 4 Punkte)

a) Sei $[1001]_2$ eine Zahl in Zweier-Komplement-Darstellung *ohne Nachkommastellen*. Um welche ganze Zahl (mit der Basis 10) handelt es sich?

b) Beweisen Sie folgendes Lemma:

Lemma: Sei $[a]_2 = [a_{n-1}a_{n-2}\dots a_0]_2$ eine *ganze* Zahl in Zweier-Komplement-Darstellung mit n Vorkommastellen und *keinen* Nachkommastellen. Dann gilt:

$$[\bar{a}]_2 + 1 = -[a]_2$$

Hierbei sei $[\bar{a}]_2$ die Zahl im Zweier-Komplement, die aus $[a]_2$ durch Invertieren aller Bits hervorgeht.

Abgesehen von der geometrischen Summenformel sollen keine Sätze aus der Vorlesung ohne Beweis benutzt werden.

Lösung:

a) $2^0 - 2^3 = -7$

$$\begin{aligned} [\bar{a}_{n-1}, \bar{a}_{n-2}, \dots, \bar{a}_0]_2 + 1 &= (\sum_{i=0}^{n-2} \bar{a}_i 2^i - \bar{a}_{n-1} 2^{n-1}) + 2^0 \\ &= -(1 - a_{n-1}) 2^{n-1} + 2^0 + \sum_{i=0}^{n-2} (1 - a_i) 2^i \\ b) &= -2^{n-1} + 2^{n-1} a_{n-1} + 2^0 + \sum_{i=0}^{n-2} 2^i - \sum_{i=0}^{n-2} a_i 2^i \\ &= -(\sum_{i=0}^{n-2} a_i 2^i - 2^{n-1} a_{n-1}) - 2^{n-1} + 2^0 + \sum_{i=0}^{n-2} 2^i \\ &= -[a]_2 - 2^{n-1} + 2^0 + \sum_{i=0}^{n-2} 2^i \\ &= -[a]_2 \end{aligned}$$

Korrekte Angabe der Definition der Zweier-Komplement-Darstellung [1]

Wenn jemand das für den allgemeinen Fall mit Nachkommastellen macht [-4]

Aufgabe 4 (3 Punkte)

Geben Sie eine einfache Methode an zum Umwandeln einer natürlichen Zahl aus Binärdarstellung in Hexadezimaldarstellung und umgekehrt ohne den Zwischenschritt der Umwandlung in das Dezimalsystem.

Lösung:

Korrekturhinweis: Es handelt sich hier um eine eher argumentative Lösung. Es ist deswegen nur wichtig, dass es kein Fehler in der Argumentation auftritt.

- VonDualAufHexadezimal
Input: Dualnummer[n] { Array von $0 \dots n - 1$ Binären Zahlen, Dualnummer[0] ist die kleinste Ziffer }
 $m = \text{ceil}(n/4)$
Hexnummer[m] {Array von $0 \dots m - 1$ Hexadezimalen Zahlen mit $b = 16$, Hexnummer[0] ist die kleinste Ziffer}
for $i = 0$; $i < n$; $++i$ **do**
 Hexnummer[$n \bmod 4$] = Hexnummer[$n \bmod 4$] + Dualnummer[n] · $2^{n \bmod 4}$
end for
return Hexnummer

- Hierfür verwenden wir eine Look-Up Tabelle HexTabelle[16], die für jede Hexadezimale Ziffer eine vierstellige Dualnummer DualArray[4] ergibt, die wie ein Array gesehen werden kann. HexTabelle ist dann ein Array von

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
Arrays 8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

VonHexadezimalAufDual

Input: Hexnummer(n) {Array von $0 \dots n - 1$ Hexadezimalen Zahlen mit $b = 16$, „[i]“ ist eine Funktion, die die i. Ziffer vom Array auszieht, Hexnummer[0] ist die kleinste Ziffer der Nummer in der Tabellenzeile}

$m = n \cdot 4$

Dualnummer(m) {Array von $0 \dots m - 1$ Binären Zahlen, Dualnummer[0] ist die kleinste Ziffer}

for $i = 0$; $i < n$; $++i$ **do**

for $j = 0$; $j < 4$; $++j$ **do**

 Dualnummer[$i \cdot 4 + j$] = HexTabelle[Hexnummer[i]][j]

end for

end for

return Dualnummer

Aufgabe 5 (4 Punkte)

Konvertieren Sie die Zahlen in Betrag & Vorzeichen, Einerkomplement und Zweierkomplement Darstellung.

-234_{10} 84_{10} $FCB1_{16}$ -1_{10} -10_{16} -654_8 $-FF_{16}$ 963_{10} -255_{10}

Lösung:

Input	B&V	1-er Kompl.	2-er Kompl.
-234_{10}	1 1110 1010	1 0001 0101	1 0001 0110
84_{10}	0101 0100	0101 0100	0101 0100
$F C B 1_{16}$	0 1111 1100 1011 0001	0 1111 1100 1011 0001	0 1111 1100 1011 0001
-1_{10}	11	10	11
-10_{16}	11 0000	10 1111	11 0000
-654_8	11 1010 1100	10 0101 0011	10 0101 0100
$-F F 1_6$	1 1111 1111	1 0000 0000	1 0000 0001
963_{10}	011 1100 0011	011 1100 0011	011 1100 0011
-255_{10}	1 1111 1111	1 0000 0000	1 0000 0001

Abgabe: 15. November 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 15. November 2013

Technische Informatik

Übungsblatt 3

Aufgabe 1 (4 Punkte)

Sei $y \in \mathbb{B}^{24}$. Dann heißt $\text{sext}(y) := y_{23}^8 y$ sign extension von y . Beweisen Sie, dass $[y] = [\text{sext}(y)]$ gilt.

Aufgabe 2 (1 + 3 Punkte)

Beim Entwurf des Instruktionssatzes eines neuen Rechners wird die Befehlsbreite auf 32 Bit festgelegt.

- Wieviele verschiedene Befehle können durch diese 32 Bit prinzipiell kodiert werden, wenn jeder Befehl einen 20-Bit-Parameter beinhaltet?
- Zur Realisierung eines MOVE-Befehls, der den Inhalt des Registers S in das Register D schreibt, werden bei der Befehlskodierung zwei Bit-Blöcke mit je 3 Bit für die Angabe von S und D reserviert, wie nachfolgende Abbildung beispielhaft veranschaulicht.

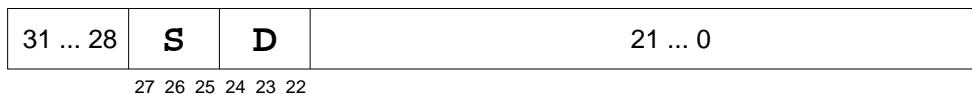


Abbildung 1: Beispielhafte Aufteilung der Datenleitungen für den MOVE-Befehl

Des Weiteren kann man annehmen, dass S und D immer unterschiedlich sind (d.h. die Befehle „MOVE S S“ bzw. „MOVE D D“ kommen nie vor). Außerdem sind die restlichen Bits (31-28 und 21-0 in der Abbildung) frei wählbar. Wieviele gültige Möglichkeiten gibt es dann für den MOVE-Befehl?

Aufgabe 3 (3 + 2 + 2 + 1 Punkte)

Gegeben sei der Schaltkreis SK_1 wie folgt:

$SK_1 := (X_3, G, typ, IN, Y)$, wobei

$$X_3 = (x_1, x_2, x_3)$$

$$Y_2 = (v_1, v_6)$$

$$G = (V, E)$$

$$V = \{x_1, x_2, x_3\} \cup \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\} \cup \{0, 1\}$$

$$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}\}$$

Die Abbildungen Q , Z , typ und IN sind durch die beiden folgenden Funktionstabellen gegeben:

$e_i \in E$	$Q(e_i)$	$Z(e_i)$
e_1	v_8	v_1
e_2	v_2	v_3
e_3	v_4	v_6
e_4	v_3	v_1
e_5	v_7	v_8
e_6	x_1	v_2
e_7	x_3	v_7
e_8	v_5	v_3
e_9	x_1	v_5
e_{10}	x_2	v_4
e_{11}	x_2	v_5
e_{12}	x_3	v_2
e_{13}	x_2	v_7
e_{14}	v_2	v_8
e_{15}	x_1	v_4
e_{16}	x_3	v_6

$v_i \in V$	$IN(v_i)$	$typ(v_i)$
v_1	(e_4, e_1)	AND_2
v_2	(e_6, e_{12})	OR_2
v_3	(e_8, e_2)	AND_2
v_4	(e_{15}, e_{10})	XOR_2
v_5	(e_9, e_{11})	OR_2
v_6	(e_3, e_{16})	XOR_2
v_7	(e_{13}, e_7)	OR_2
v_8	(e_{14}, e_5)	AND_2

- a) Zeichnen Sie SK_1 .
- b) Eine topologische Sortierung eines azyklischen Graphen $G = (V, E)$ ist eine bijektive Abbildung $topsort : V \rightarrow \{1, \dots, |V|\}$ mit folgender Eigenschaft: $\forall e \in E : topsort(Q(e)) < topsort(Z(e))$. Unter einer topologischen Sortierung eines kombinatorischen Schaltkreises versteht man eine topologische Sortierung des azyklischen Graphen, der dem Schaltkreis zugrunde liegt. Geben Sie für SK_1 eine topologische Sortierung an.
- c) Bestimmen Sie die Kosten sowie die Tiefe von SK_1 , wobei die Kosten sich aus der Anzahl der benutzen Gatter ergibt und die Tiefe entspricht der Anzahl Gatter, die auf dem längsten Pfad von G liegen.
- d) Geben Sie eine Simulation für den Eingangsvektor $(0, 1, 0)$ an. Werten Sie dazu die Gatter in der durch Ihre topologische Sortierung gegebenen Reihefolge aus.

Aufgabe 4 (3 + 5 Punkte)

Wer von Ihnen die älteren Folgen von „Star Trek“ (dt. „Raumschiff Enterprise“) gesehen hat, dem ist vielleicht aufgefallen, daß der Hauptcomputer der U.S.S. Enterprise nicht wie üblich auf binärer, sondern auf *trinärer* Logik basiert (also nicht nur 0 und 1, sondern einen zusätzlichen dritten Wert verwendet). Allerdings kann man sehen, daß auf diesem Computer die Prinzipien der Booleschen Algebra nicht anwendbar sein können.

Dazu benötigen sie zunächst zwei Korrolare, die Sie ohne Beweis verwenden dürfen:

- **Korrolar 1:**

Für jede Boolesche Algebra $(M, \cdot, +, \bar{})$ gilt:

$$x + \bar{x} = \mathbf{1}, \quad x \cdot \bar{x} = \mathbf{0} \quad \forall x \in M$$

Wobei $\mathbf{0}$ das 0-Element und $\mathbf{1}$ das 1-Element der Booleschen Algebra sind.

- **Korrolar 2:** (Komplement neutraler Elemente)

Für jede Boolesche Algebra $(M, \cdot, +, \bar{})$ gilt

$$\mathbf{0} = \bar{\mathbf{1}}, \quad \mathbf{1} = \bar{\mathbf{0}}$$

$\mathbf{0}$ und $\mathbf{1}$ sind wiederum das 0- bzw. 1-Element der Booleschen Algebra.

Beweisen Sie nun erst formal das folgende Lemma und verwenden Sie dieses dann zum Beweis des folgenden Satzes:

a) **Lemma:** Für jede Boolesche Algebra $(M, \cdot, +, \bar{})$ mit $|M| > 1$ gilt $\mathbf{0} \neq \mathbf{1}$.

b) **Satz:** Es existiert keine Boolesche Algebra $(M, \cdot, +, \bar{})$ mit $|M| = 3$.

Hinweis: Beweisen Sie zunächst, daß es ein $a \in M$ gibt mit $a = \bar{a}$.

Sie dürfen sowohl die in der Vorlesung für Boolesche Algebren definierten Axiome, die in der Vorlesung angegebenen (teils unbewiesenen) weiteren Regeln für Boolesche Algebren und die beiden Korrolare verwenden.

Geben Sie für in allen Teilaufgaben für jeden Beweisschritt an, welche Regel verwendet wurde.

Abgabe: 22. November 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 15. November 2013

Technische Informatik

Musterlösung zu Übungsblatt 3

Aufgabe 1 (4 Punkte)

Sei $y \in \mathbb{B}^{24}$. Dann heißt $\text{sext}(y) := y_{23}^8 y$ sign extension von y . Beweisen Sie, dass $[y] = [\text{sext}(y)]$ gilt.

Lösung:

Lemma: Sei $a \in \mathbb{B}^n$, $a = a_{n-1} \dots a_0$. Dann gilt $[a] = [a_{n-1}a]$.
Beweis:

$$[a_{n-1}a] = -a_{n-1} \cdot 2^n + \sum_{i=0}^{n-1} a_i \cdot 2^i = -a_{n-1} \cdot 2^n + \left(a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i \right) = -a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i = [a]$$

Damit: $[y] = [y_{23}^1 y] = [y_{23}^2 y] = \dots = [y_{23}^8 y] = [\text{sext}(y)]$.

Punkte: Rechnung mit der Interpretationsfunktion [1,5]; Wiederanwendung bis $[y] = [\text{sext}(y)]$ [0,5]

Aufgabe 2 (1 + 3 Punkte)

Beim Entwurf des Instruktionssatzes eines neuen Rechners wird die Befehlsbreite auf 32 Bit festgelegt.

- a) Wieviele verschiedene Befehle können durch diese 32 Bit prinzipiell kodiert werden, wenn jeder Befehl einen 20-Bit-Parameter beinhaltet?
- b) Zur Realisierung eines MOVE-Befehls, der den Inhalt des Registers S in das Register D schreibt, werden bei der Befehlskodierung zwei Bit-Blöcke mit je 3 Bit für die Angabe von S und D reserviert, wie nachfolgende Abbildung beispielhaft veranschaulicht.

31 ... 28	S	D	21 ... 0
27 26 25 24 23 22			

Abbildung 1: Beispielhafte Aufteilung der Datenleitungen für den MOVE-Befehl

Desweiteren kann man annehmen, dass **S** und **D** immer unterschiedlich sind (d.h. die Befehle „MOVE **S S“ bzw. „MOVE **D D“ kommen nie vor). Ausserdem sind die restlichen Bits (31-28 und 21-0 in der Abbildung) frei wählbar. Wieviele gültige Möglichkeiten gibt es dann für den MOVE-Befehl?****

Lösung:

- a) Natürlich kommen alle 2^{32-20} Belegungen für eine Befehlskodierung in Betracht, so dass man also 2^{12} Befehle kodieren kann.
- b) Da nur auf den 6 Leitungen für die Register **S** und **D** Einschränkungen bestehen, sind die restlichen Leitungen frei wählbar (schon mal 2^{26} Möglichkeiten). Da **S** und **D** unterschiedlich sein müssen, gibt es nur folgende gültigen Paare:

S	D	S	D	usw.
000	001	001	000	
000	010	001	010	
000	011	001	011	
000	100	001	100	
000	101	001	101	
000	110	001	110	
000	111	001	111	

Allgemein gibt es bei einer Bitbreite n für die Register **S** und **D** $2^n \cdot (2^n - 1)$ Möglichkeiten, in dem konkreten Fall also $8 \cdot 7 = 56$ Möglichkeiten. Zusammen mit den anderen Leitungen ergeben sich also

$$2^{26} \cdot 2^3 \cdot (2^3 - 1) = 2^{32} - 2^{29} = 7 \cdot 2^{29}$$

Möglichkeiten für den MOVE-Befehl.

Aufgabe 3 (3 + 2 + 2 + 1 Punkte)

Gegeben sei der Schaltkreis SK_1 wie folgt:

$SK_1 := (X_3, G, typ, IN, Y)$, wobei

$$X_3 = (x_1, x_2, x_3)$$

$$Y_2 = (v_1, v_6)$$

$$G = (V, E)$$

$$V = \{x_1, x_2, x_3\} \cup \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\} \cup \{0, 1\}$$

$$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}\}$$

Die Abbildungen Q , Z , typ und IN sind durch die beiden folgenden Funktionstabellen gegeben:

$e_i \in E$	$Q(e_i)$	$Z(e_i)$
e_1	v_8	v_1
e_2	v_2	v_3
e_3	v_4	v_6
e_4	v_3	v_1
e_5	v_7	v_8
e_6	x_1	v_2
e_7	x_3	v_7
e_8	v_5	v_3
e_9	x_1	v_5
e_{10}	x_2	v_4
e_{11}	x_2	v_5
e_{12}	x_3	v_2
e_{13}	x_2	v_7
e_{14}	v_2	v_8
e_{15}	x_1	v_4
e_{16}	x_3	v_6

$v_i \in V$	$IN(v_i)$	$typ(v_i)$
v_1	(e_4, e_1)	AND_2
v_2	(e_6, e_{12})	OR_2
v_3	(e_8, e_2)	AND_2
v_4	(e_{15}, e_{10})	XOR_2
v_5	(e_9, e_{11})	OR_2
v_6	(e_3, e_{16})	XOR_2
v_7	(e_{13}, e_7)	OR_2
v_8	(e_{14}, e_5)	AND_2

- a) Zeichnen Sie SK_1 .
- b) Eine topologische Sortierung eines azyklischen Graphen $G = (V, E)$ ist eine bijektive Abbildung $topsort : V \rightarrow \{1, \dots, |V|\}$ mit folgender Eigenschaft: $\forall e \in E : topsort(Q(e)) < topsort(Z(e))$. Unter einer topologischen Sortierung eines kombinatorischen Schaltkreises versteht man eine topologische Sortierung des azyklischen Graphen, der dem Schaltkreis zugrunde liegt. Geben Sie für SK_1 eine topologische Sortierung an.
- c) Bestimmen Sie die Kosten sowie die Tiefe von SK_1 , wobei die Kosten sich aus der Anzahl der benutzen Gatter ergibt und die Tiefe entspricht der Anzahl Gatter, die auf dem längsten Pfad von G liegen.
- d) Geben Sie eine Simulation für den Eingangsvektor $(0, 1, 0)$ an. Werten Sie dazu die Gatter in der durch Ihre topologische Sortierung gegebenen Reihefolge aus.

Lösung:

- a) Skizze des Schaltkreises SK_1 : siehe Abbildung 2.

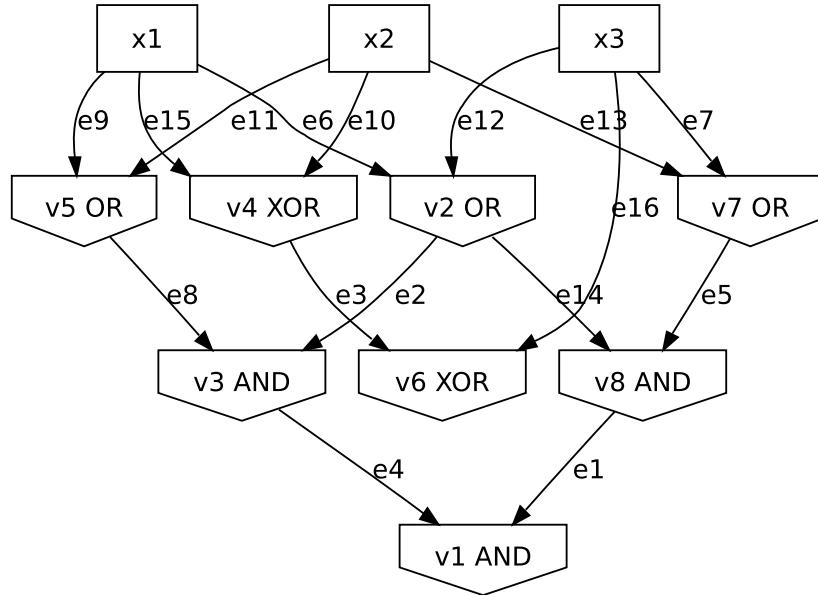


Abbildung 2: sk

- b) Etwa

$$tsort = \{(x_1 \mapsto 1), (x_2 \mapsto 2), (x_3 \mapsto 3), \\ (v_5 \mapsto 4), (v_2 \mapsto 5), (v_7 \mapsto 6), (v_4 \mapsto 7), \\ (v_3 \mapsto 8), (v_8 \mapsto 9), (v_6 \mapsto 10), (v_1 \mapsto 11)\}$$

- c) Kosten 8, Tiefe 3.

Folgefehler aus a) sind ok. Jeweils 1 Punkt.

Gatterausgang=Output		
x_1	=	0
x_2	=	1
x_3	=	0
v_5	=	1
v_2	=	0
v_7	=	1
v_4	=	1
v_3	=	0
v_8	=	0
v_6	=	1
v_1	=	0

Aufgabe 4 (3 + 5 Punkte)

Wer von Ihnen die älteren Folgen von „Star Trek“ (dt. „Raumschiff Enterprise“) gesehen hat, dem ist vielleicht aufgefallen, daß der Hauptcomputer der U.S.S. Enterprise nicht wie üblich auf binärer,

sondern auf *trinärer* Logik basiert (also nicht nur 0 und 1, sondern einen zusätzlichen dritten Wert verwendet). Allerdings kann man sehen, daß auf diesem Computer die Prinzipien der Booleschen Algebra nicht anwendbar sein können.

Dazu benötigen sie zunächst zwei Korrolare, die Sie ohne Beweis verwenden dürfen:

- **Korrolar 1:**

Für jede Boolesche Algebra $(M, \cdot, +, \bar{})$ gilt:

$$x + \bar{x} = \mathbf{1}, \quad x \cdot \bar{x} = \mathbf{0} \quad \forall x \in M$$

Wobei $\mathbf{0}$ das 0-Element und $\mathbf{1}$ das 1-Element der Booleschen Algebra sind.

- **Korrolar 2:** (Komplement neutraler Elemente)

Für jede Boolesche Algebra $(M, \cdot, +, \bar{})$ gilt

$$\mathbf{0} = \bar{\mathbf{1}}, \quad \mathbf{1} = \bar{\mathbf{0}}$$

$\mathbf{0}$ und $\mathbf{1}$ sind wiederum das 0- bzw. 1-Element der Booleschen Algebra.

Beweisen Sie nun erst formal das folgende Lemma und verwenden Sie dieses dann zum Beweis des folgenden Satzes:

a) **Lemma:** Für jede Boolesche Algebra $(M, \cdot, +, \bar{})$ mit $|M| > 1$ gilt $\mathbf{0} \neq \mathbf{1}$.

b) **Satz:** Es existiert keine Boolesche Algebra $(M, \cdot, +, \bar{})$ mit $|M| = 3$.

Hinweis: Beweisen Sie zunächst, daß es ein $a \in M$ gibt mit $a = \bar{a}$.

Sie dürfen sowohl die in der Vorlesung für Boolesche Algebren definierten Axiome, die in der Vorlesung angegebenen (teils unbewiesenen) weiteren Regeln für Boolesche Algebren und die beiden Korrolare verwenden.

Geben Sie für in allen Teilaufgaben für jeden Beweisschritt an, welche Regel verwendet wurde.

Lösung:

Punkte (für alle Teilaufgaben): Mit der Angabe der Regeln bei a) und b) nicht so streng sein.

a) Beweis durch Widerspruch:

Sei $(M, \cdot, +, \bar{})$ mit $|M| > 1$ eine boolesche Algebra und zusätzlich gelte für diese boolesche Algebra $\mathbf{0} = \mathbf{1}$.

Wegen $|M| > 1$ und $\mathbf{0} = \mathbf{1}$ gibt es ein $a \in M$ mit $a \neq \mathbf{0}$ und $a \neq \mathbf{1}$.

$$a \stackrel{(\text{Neutr. El.})}{=} a + \mathbf{0} \stackrel{\mathbf{0}=\mathbf{1}}{=} a + \mathbf{1} \stackrel{(\text{Neutr. El.})}{=} \mathbf{1}. \quad \text{Widerspruch zu } a \neq \mathbf{1}.$$

b) Zwei Beweise:

- Beweis von Prof. Scholl durch Widerspruch: Sei $(M, \cdot, +, \bar{})$ mit $|M| = 3$ eine boolesche Algebra, oBdA sei $M = \{2, 3, 4\}$, $3 = \mathbf{0}$ und $4 = \mathbf{1}$ (müssen verschieden sein wg. Aufgabenteil a).

Aus Eindeutigkeit der Neutralen Elemente folgt $2 \neq \mathbf{0}$ (*) und $2 \neq \mathbf{1}$ (**).

$$\left. \begin{array}{l} \bar{2} = \mathbf{1} \stackrel{\text{Korrolar 2}}{\Rightarrow} 2 = \mathbf{0} \quad \text{Widerspruch zu (*)} \\ \bar{2} = \mathbf{0} \stackrel{\text{Korrolar 2}}{\Rightarrow} 2 = \mathbf{1} \quad \text{Widerspruch zu (**)} \end{array} \right\} \Rightarrow \bar{2} = 2$$

$$2 \stackrel{\text{Idempotenz}}{=} 2 + 2 \stackrel{\bar{2}=2}{=} 2 + \bar{2} \stackrel{\text{Korrolar 1}}{=} \mathbf{1}. \quad \text{Widerspruch}$$

- Beweis von Tobias durch Widerspruch:

Sei $(M, \cdot, +, \bar{})$ mit $|M| = 3$ eine boolesche Algebra, oBdA sei $M = \{2, 3, 4\}$.

Behauptung: $\exists a \in M : \bar{a} = a$, Beweis durch Betrachten aller möglichen Fälle für $\bar{2}$:

- Fall 1: $\bar{2} = 2$ Behauptung erfüllt.
- Fall 2: $\bar{2} = 3$. Wegen $\bar{\bar{2}} = 2$ (doppeltes Komplement) folgt $\bar{3} = 2$.
 - * $\bar{4} \neq 2$, da sonst $\bar{\bar{4}} = \bar{2} = 3$ (Widerspruch zu doppeltes Komplement),
 - * $\bar{4} \neq 3$, da sonst $\bar{\bar{4}} = \bar{3} = 2$ (Widerspruch zu doppeltes Komplement),
 - $\Rightarrow \bar{4} = 4$. Behauptung erfüllt.
- Fall 3: $\bar{2} = 4$. Analog zu Fall 2 gilt $\bar{3} = 3$. Behauptung erfüllt.

Sei oBdA $\bar{2} = 2$.

Es folgt:

- $2 \stackrel{\text{Idempotenz}}{=} 2 + 2 \stackrel{\bar{2}=2}{=} 2 + \bar{2} \stackrel{\text{Korollar } 1}{=} \mathbf{1}$.
- $2 \stackrel{\text{Idempotenz}}{=} 2 \cdot 2 \stackrel{\bar{2}=2}{=} 2 \cdot \bar{2} \stackrel{\text{Korollar } 1}{=} \mathbf{0}$.

Widerspruch zu Aufgabenteil a).

□

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 22. November 2013

Technische Informatik Übungsblatt 4

Aufgabe 1 (4 Punkte)

Beweisen Sie folgendes Lemma der Vorlesung:

Zu jedem Booleschen Ausdruck $e \in BE(X_n)$ gibt es einen Schaltkreis $SK = (\vec{X}_n, G, typ, IN, \vec{Y}_1)$, so dass gilt: $\psi(e) = f_{SK}$.

Hinweis: Induktion über die Struktur Boolescher Ausdrücke.

Aufgabe 2 (3 + 3 Punkte)

In der Vorlesung wurde die Bibliothek der Standardzellen vorgestellt.

- a) Geben Sie zu den Funktionen and_2 , or_2 und not jeweils einen Schaltkreis über der Zellenbibliothek $\{nand_2\}$ an, der die entsprechende Funktion implementiert.
- b) Geben Sie ein kurzes Argument dafür an, dass es zu jeder Booleschen Funktion $f \in B_n$ einen Schaltkreis über der Bibliothek $BIB = \{nand_2\}$ gibt.

Aufgabe 3 (3 + 2 + 3 + 2 Punkte)

Betrachten Sie den PLA in Abbildung 1, der eine boolesche Funktion $g: \mathbb{B}^3 \rightarrow \mathbb{B}$ realisiert.

- a) Geben Sie das vom PLA dargestellte Polynom p_g zu g an.
- b) Zeigen Sie, dass das Polynom p_g *kein* Minimalpolynom von g ist.
- c) Geben Sie einen Schaltkreis $SK(g)$ an, der g realisiert.
Benutzen Sie dabei *ausschließlich* NAND- und Inverter-Gatter!
- d) Geben Sie die formale Schaltkreisdefinition $SK(g) = (X_3, G, typ, IN, Y)$ für den Schaltkreis aus Teilaufgabe c) an.

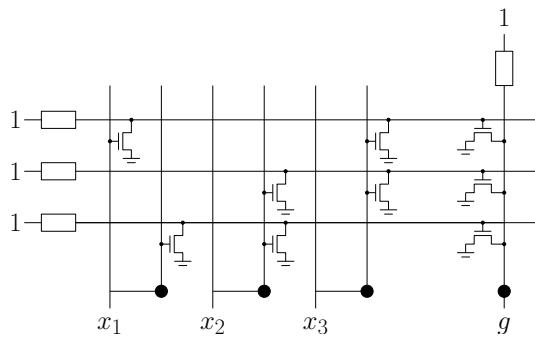


Abbildung 1: Ein PLA

(Bonus)

Aufgabe 4 (2 Punkte)

Zeigen Sie, dass die kDNF im Allgemeinen nicht die kostenminimale DNF-Darstellung ist.

Aufgabe 5 (1 + 3 Punkte)

- a) Geben Sie für die gegebene Booleschen Funktion einen Booleschen Ausdruck in der kanonischen disjunktiven Normalform an. Sie müssen den Ausdruck nicht in vollständig geklammerter Form angeben.

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

- b) Geben Sie für den Booleschen Ausdruck $e = x_1 + \neg(x_2 \cdot \neg x_3)$ die zugehörige Interpretationsfunktion $\psi : BE(x_1, x_2, x_3) \rightarrow \mathbb{B}_3$ an. Gehen Sie dabei schrittweise vor und stellen Sie $\psi(e)$ durch Disjunktion/Konjunktion/Negationen von Projektionsfunktionen dar. Berechnen Sie schließlich die Funktionstabelle von $\psi(e)$.

Abgabe: 29. November 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
 Dr. Paolo Marin

Freiburg, 22. November 2013

Technische Informatik Musterlösung zu Übungsblatt 4

Aufgabe 1 (4 Punkte)

Beweisen Sie folgendes Lemma der Vorlesung:

Zu jedem Booleschen Ausdruck $e \in BE(X_n)$ gibt es einen Schaltkreis $SK = (\vec{X}_n, G, typ, IN, \vec{Y}_1)$, so dass gilt: $\psi(e) = f_{SK}$.

Hinweis: Induktion über die Struktur Boolescher Ausdrücke.

Lösung:

Induktionsanfang:

Sei $BE_0 = \{0, 1, x_1, \dots, x_n\}$.

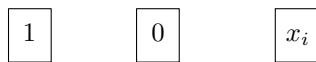


Abbildung 1: Induktionsanfang, SK_1 , SK_0 , SK_{x_i}

Induktionsvoraussetzung.

Für alle booleschen Ausdrücke $e \in BE_{i-1}$ existiert ein Schaltkreis f . Fallunterscheidung:

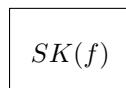


Abbildung 2: IV

i) $e = \neg f$, $f \in BE_{i-1}$

ii) $e = fg$, $f, g \in BE_{i-1}$

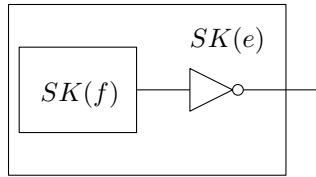


Abbildung 3: IS für *NOT*

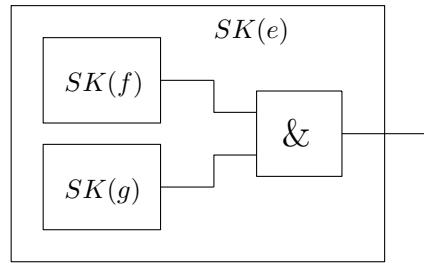


Abbildung 4: IS für *AND*

- iii) $e = f + g$, $f, g \in BE_i$
 Analog zu ii)

Punkte: IA [0.5]; IV formuliert [0.5]; auf den Rest irgendwie 2 Punkte verteilen; Ein Fall bei IS vergessen [-0.5]

Aufgabe 2 (3 + 3 Punkte)

In der Vorlesung wurde die Bibliothek der Standardzellen vorgestellt.

- Geben Sie zu den Funktionen and_2 , or_2 und not jeweils einen Schaltkreis über der Zellenbibliothek $\{nand_2\}$ an, der die entsprechende Funktion implementiert.
- Geben Sie ein kurzes Argument dafür an, dass es zu jeder Booleschen Funktion $f \in B_n$ einen Schaltkreis über der Bibliothek $BIB = \{nand_2\}$ gibt.

Lösung:

- $and_2(x, y) = nand_2(nand_2(x, y), nand_2(x, y))$
 - $or_2(x, y) = nand_2(nand_2(x, x), nand_2(y, y))$
 - $not(x) = nand_2(x, x)$
- In der Teilaufgabe a) haben sie or_2 , and_2 und not durch einen Schaltkreis mit $nand_2$ ausgedrückt. Dann braucht man doch nur noch, dass es für alle Booleschen Funktionen einen Schaltkreis über $BIB = \{and_2, or_2, not\}$ gibt (folgt aus Lemma 3 und 2 der Vorlesung) und man ersetzt die Gatter entsprechend durch Teilschaltkreise mit $nand_2$.
 (Das *NAND*-Gatter ist auch als universelles Gatter bekannt)

Aufgabe 3 (3 + 2 + 3 + 2 Punkte)

Betrachten Sie den PLA in Abbildung 5, der eine boolesche Funktion $g: \mathbb{B}^3 \rightarrow \mathbb{B}$ realisiert.

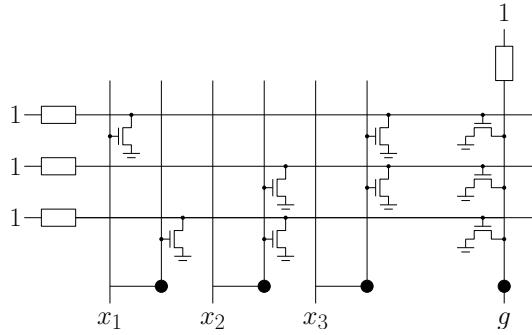


Abbildung 5: Ein PLA

- Geben Sie das vom PLA dargestellte Polynom p_g zu g an.
- Zeigen Sie, dass das Polynom p_g *kein* Minimalpolynom von g ist.
- Geben Sie einen Schaltkreis $SK(g)$ an, der g realisiert.
Benutzen Sie dabei *ausschließlich* NAND- und Inverter-Gatter!
- Geben Sie die formale Schaltkreisdefinition $SK(g) = (X_3, G, typ, IN, Y)$ für den Schaltkreis aus Teilaufgabe c) an.

Lösung:

- $p_g = \bar{x}_1x_3 + x_2x_3 + x_1x_2$ [3]
Wenn falschrum negiert, [-1.5]
- $p'_g = \bar{x}_1x_3 + x_1x_2$, $\psi(p_g) = \psi(p'_g)$. [2] Zeigen zB durch Wahrheitstabelle oder durch Consensus-Regel [1]
Folgefehler nach falschrum negiert in a) ist OK.
- Viele denkbare Lösungen. Optimal:

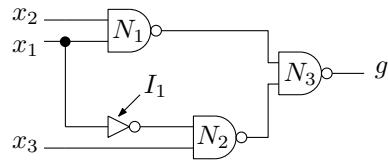


Abbildung 6: Schaltkreis für g

Punkte: Nicht ausschließlich mit NAND/Inverter, aber korrekt, noch [2]

- $SK(g) = (X_3, G, typ, IN, Y)$

$$\begin{aligned}
 X_3 &= \{x_1, x_2, x_3\} \\
 Y &= \{g\} \\
 G &= (V, E) \\
 V &= X_3 \cup \{N_1, N_2, N_3, I_1\} \cup Y \\
 E &= \{(x_1, I_1), (x_1, N_1), (x_2, N_1), (x_3, N_3), (N_1, N_3), (N_2, N_3), (N_3, g)\} \\
 typ &= \{(N_1 \mapsto \text{NAND}), (N_2 \mapsto \text{NAND}), (N_3 \mapsto \text{NAND}), (I_1 \mapsto \text{NEG})\} \\
 IN &= \{(N_1 \mapsto (x_1, x_2)), (N_2 \mapsto (I_1, x_3)), (N_3 \mapsto (N_1, N_2)), (I_1 \mapsto (x_1))\}
 \end{aligned}$$

Punkte: x_i bzw. g bei V und E vergessen, einmalig [-1.5]; Sonstige fehlende oder zu viele Angaben, jeweils [-0.5], keine Minuspunkte; es ist auch ok N_3 als Schaltkreisausgang zu definieren – entsprechend benötigt man die Kante (N_3, g) nicht mehr;

(Bonus)

Aufgabe 4 (2 Punkte)

Zeigen Sie, dass die kDNF im Allgemeinen nicht die kostenminimale DNF-Darstellung ist.

Lösung:

Dies kann man z.B. durch Angabe eines Beispiels zeigen. Sei eine boolesche Funktion f gegeben:

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$\begin{aligned} ON(f) &= \{001, 110, 111\} \\ kDNF(f) &= \bar{x} \cdot \bar{y} \cdot z + x \cdot y \cdot \bar{z} + x \cdot y \cdot z \end{aligned}$$

Die kDNF von f besteht also aus 3 Monomen.

Es gibt aber weitere äquivalente DNF-Realisierungen von f , z.B.

$$p = \bar{x} \cdot \bar{y} \cdot z + x \cdot y$$

mit 2 Monomen.

Damit ist gezeigt, dass die kDNF nicht kostenminimal ist.

Aufgabe 5 (1 + 3 Punkte)

- a) Geben Sie für die gegebene Booleschen Funktion einen Booleschen Ausdruck in der kanonischen disjunktiven Normalform an. Sie müssen den Ausdruck nicht in vollständig geklammerter Form angeben.

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

- b) Geben Sie für den Booleschen Ausdruck $e = x_1 + \neg(x_2 \cdot \neg x_3)$ die zugehörige Interpretationsfunktion $\psi : BE(x_1, x_2, x_3) \rightarrow \mathbb{B}_3$ an. Gehen Sie dabei schrittweise vor und stellen Sie $\psi(e)$ durch Disjunktion/Konjunktion/Negationen von Projektionsfunktionen dar. Berechnen Sie schließlich die Funktionstabelle von $\psi(e)$.

Lösung:

a) $\overline{x_1x_2x_3} + \overline{x_1}x_2\overline{x_3} + x_1\overline{x_2}\overline{x_3} + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3}$

b)

$$\begin{aligned}
 & \psi(x_1 + \neg(x_2 \cdot \neg x_3)) && (1) \\
 &= (\psi(x_1) + \psi(\neg(x_2 \cdot \neg x_3))) && (2) \\
 &= (\psi(x_1) + (\neg(\psi(x_2 \cdot \neg x_3)))) && (3) \\
 &= (\psi(x_1) + (\neg(\psi(x_2) \cdot \psi(\neg x_3)))) && (4) \\
 &= (\psi(x_1) + (\neg(\psi(x_2) \cdot (\neg\psi(x_3))))) && (5)
 \end{aligned}$$

α	$(\psi(x_1) + (\neg(\psi(x_2) \cdot (\neg\psi(x_3)))))(\alpha)$
0 0 0	1
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

Abgabe: 29. November 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 29. November 2013

Technische Informatik

Übungsblatt 5

Aufgabe 1 (3 + 1 + 3 + 1 + 1 + 2 Punkte)

Die Funktion $f: \mathbb{B}^4 \rightarrow \mathbb{B}$ sei durch ihre ON-Menge gegeben:

$$ON(f) = \{0000, 0001, 0100, 0101, 0110, 0111, 1000, 1010, 1100, 1110, 1111\}$$

- a) Bestimmen Sie durch Anwendung des Quine-McCluskey-Algorithmus die Primimplikantenmenge von f . Geben Sie alle Zwischenschritte, d.h. alle Mengen L_i^M und $Prim$ an.

Hinweis: Sie dürfen in dieser Aufgabe die abkürzende Schreibweise für Monome verwenden (z.B. statt „ $\bar{x}_1x_2x_4$ “ „01-1“)

- b) Zeichnen Sie in einem Hypercube die ON-Menge von f ein. Markieren Sie zusätzlich im Hypercube alle Primimplikanten, die Sie durch den Quine-McCluskey-Algorithmus erhalten haben. Hinweis:

Wenn Ihnen im Quine-McCluskey-Algorithmus kein Fehler unterlaufen sein, sollten die Teilwürfel, die durch die Primimplikanten aufgespannt werden, jeweils maximal sein, d.h. es ist nicht möglich, die Würfel zu vergrößern, ohne die ON-Menge zu verlassen. Gleichzeitig sollten alle Ecken, die in der ON-Menge vorkommen auch in mindestens einem der Teilwürfel vorkommen.

- c) Erstellen Sie eine Primimplikantentafel zu f .

Reduzieren Sie die Primimplikantentafel nun solange mit der *zweiten Reduktionsregel* (Entfernen von Spalten), bis die Regel nicht mehr angewandt werden kann. Geben Sie dabei bei jedem Reduktionsschritt an, über welche Spalten Sie argumentieren.

Hinweis: Es ist nicht notwendig, in jedem Schritt auch eine neue Primimplikantentafel zu erstellen; streichen Sie in der Tafel einfach die entsprechenden Spalten durch.

- d) Wenden Sie die *erste Reduktionsregel* erst dann auf die Primimplikantentafel an, wenn die zweite Reduktionsregel nicht mehr angewandt werden kann; geben Sie auch hier die einzelnen verwendeten Reduktionsschritte an.
- e) Geben Sie das Minimalpolynom zu f an.
- f) Bestimmen Sie die Kosten (($cost_1, cost_2$) gemäß der Vorlesung) des (i) Minimalpolynoms und (ii) des vollständigen Polynoms, das aus allen Primimplikanten der Funktion besteht.

Aufgabe 2 (4 Punkte)

Beim Zusammenfassen zweier Implikanten nach der Methode von Quine-McCluskey resultiert ebenfalls ein Implikant der gegebenen Funktion f .

Diese Aussage lässt sich für einen beliebigen Implikanten i , der x_j nicht enthält, durch folgende Boolesche Formel ausdrücken:

$$\left(((x_j \wedge i) \rightarrow f) \wedge ((\neg x_j \wedge i) \rightarrow f) \right) = (i \rightarrow f)$$

Führen Sie den Beweis dieser Formel ausschließlich mittels Umformungen nach den Gesetzen der Booleschen Algebra durch. Sie dürfen neben den Axiomen alle Folgerungen aus der Vorlesung benutzen.

Außerdem sei der Operator \rightarrow wie folgt definiert:

$$(a \rightarrow b) = (\neg a \vee b)$$

Geben Sie bei jeder Ihrer Umformungen an, welche Regeln/Axiome der Booleschen Algebra Sie verwendet haben.

Aufgabe 3 (5 + 4 Punkte)

Die Funktion $g : \mathbb{B}^4 \rightarrow \mathbb{B}$ sei gegeben durch die Menge ihrer Primimplikanten $Prim(g)$ mit

$$Prim(g) = \{\bar{x}_1x_2x_4, x_2\bar{x}_3x_4, x_1\bar{x}_3x_4, x_1\bar{x}_2x_4, \bar{x}_2x_3x_4, \bar{x}_1x_3x_4\}$$

- Bestimmen Sie *alle* Minimalpolynome von g mit Hilfe der Methode von Petrick. Sie können davon ausgehen, dass alle Primimplikanten die gleichen Kosten haben.
- Gibt es eine Boolesche Funktion mit einem Minimalpolynom, das nicht ausschließlich aus Primimplikanten besteht? Beweisen Sie Ihre Aussage.

Abgabe: 6. Dezember 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 29. November 2013

Technische Informatik Musterlösung zu Übungsblatt 5

Aufgabe 1 (3 + 1 + 3 + 1 + 1 + 2 Punkte)

Die Funktion $f: \mathbb{B}^4 \rightarrow \mathbb{B}$ sei durch ihre ON-Menge gegeben:

$$ON(f) = \{0000, 0001, 0100, 0101, 0110, 0111, 1000, 1010, 1100, 1110, 1111\}$$

- a) Bestimmen Sie durch Anwendung des Quine-McCluskey-Algorithmus die Primimplikantenmenge von f . Geben Sie alle Zwischenschritte, d.h. alle Mengen L_i^M und $Prim$ an.

Hinweis: Sie dürfen in dieser Aufgabe die abkürzende Schreibweise für Monome verwenden (z.B. statt „ $\bar{x}_1x_2x_4$ “ „01-1“)

- b) Zeichnen Sie in einem Hypercube die ON-Menge von f ein. Markieren Sie zusätzlich im Hypercube alle Primimplikanten, die Sie durch den Quine-McCluskey-Algorithmus erhalten haben. Hinweis:

Wenn Ihnen im Quine-McCluskey-Algorithmus kein Fehler unterlaufen sein, sollten die Teilwürfel, die durch die Primimplikanten aufgespannt werden, jeweils maximal sein, d.h. es ist nicht möglich, die Würfel zu vergrößern, ohne die ON-Menge zu verlassen. Gleichzeitig sollten alle Ecken, die in der ON-Menge vorkommen auch in mindestens einem der Teilwürfel vorkommen.

- c) Erstellen Sie eine Primimplikantentafel zu f .

Reduzieren Sie die Primimplikantentafel nun solange mit der *zweiten Reduktionsregel* (Entfernen von Spalten), bis die Regel nicht mehr angewandt werden kann. Geben Sie dabei bei jedem Reduktionsschritt an, über welche Spalten Sie argumentieren.

Hinweis: Es ist nicht notwendig, in jedem Schritt auch eine neue Primimplikantentafel zu erstellen; streichen Sie in der Tafel einfach die entsprechenden Spalten durch.

- d) Wenden Sie die *erste Reduktionsregel* erst dann auf die Primimplikantentafel an, wenn die zweite Reduktionsregel nicht mehr angewandt werden kann; geben Sie auch hier die einzelnen verwendeten Reduktionsschritte an.
- e) Geben Sie das Minimalpolynom zu f an.
- f) Bestimmen Sie die Kosten (($cost_1, cost_2$) gemäß der Vorlesung) des (i) Minimalpolynoms und (ii) des vollständigen Polynoms, das aus allen Primimplikanten der Funktion besteht.

Lösung:

- a) pro falschem oder vergessenen Implikanten in L_i^M oder $Prim[-1]$. Folgefehler sind ok. [-2] wenn L_i nicht nach Variablenmengen geteilt wurde. [-1] wenn PI in $Prim$ zu früh auftauchen.

$$L_0^{\{x_1, x_2, x_3, x_4\}}(f) = \{0000, 0001, 0100, 0101, 0110, 0111, 1000, 1010, 1100, 1110, 1111\}$$

$$Prim(f) = \{\}$$

$$L_1^{\{x_1, x_2, x_3\}}(f) = \{000-, 010-, 011-, 111-\}$$

$$L_1^{\{x_1, x_2, x_4\}}(f) = \{01-0, 01-1, 10-0, 11-0\}$$

$$L_1^{\{x_1, x_3, x_4\}}(f) = \{0-00, 0-01, 1-00, 1-10\}$$

$$L_1^{\{x_2, x_3, x_4\}}(f) = \{-000, -100, -110, -111\}$$

$$Prim(f) = \{\}$$

$$L_2^{\{x_1, x_2\}} = \{01--\}$$

$$L_2^{\{x_1, x_3\}} = \{0-0-\}$$

$$L_2^{\{x_1, x_4\}} = \{1--0\}$$

$$L_2^{\{x_2, x_3\}} = \{-11-\}$$

$$L_2^{\{x_2, x_4\}} = \{-1-0\}$$

$$L_2^{\{x_3, x_4\}} = \{--00\}$$

$$Prim(f) = \{\}$$

$$L_3^{\{x_1\}} = \{\}$$

$$L_3^{\{x_2\}} = \{\}$$

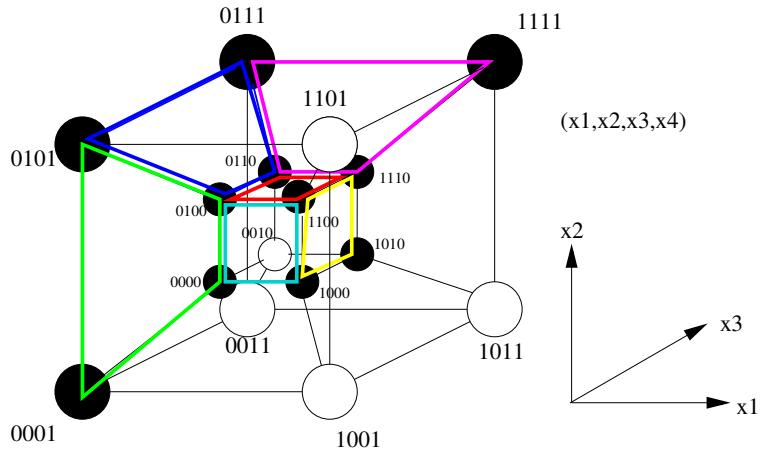
$$L_3^{\{x_3\}} = \{\}$$

$$L_3^{\{x_4\}} = \{\}$$

$$Prim(f) = \{01--, 0-0-, 1--0, -11-, -1-0, --00\}$$

$$\bigcup_M L_3^M(f) = \{\} \Rightarrow \text{Abbruch der Schleife, } Prim(f) \text{ wird zurückgegeben}$$

- b) ON-Menge korrekt eingetragen [1], falscher Teilwürfel [-0,5], Folgefehler aus a) sind nicht ok, wenn man sie im Hypercube hätte erkennen können.



[01--] [0-0-] [1--0] [-11-] [-1-0] [--00]

- c) PIT: pro falscher Eintrag [-0,5], pro falscher Regel [-0,5], nicht an Vorgaben gehalten (erst Regel 2, dann Regel 1) einmal [-1]. Folgefehler sind ok.

	0000	0001	0100	0101	0110	0111	1000	1010	1100	1110	1111
--00	1		1				1		1		
-1-0			1		1				1	1	
-11-					1	1				1	1
1--0							1	1	1	1	
0-0-	1	1	1	1							
01--			1	1	1	1					

0000, 0100 und 0101 dominieren 0001 \Rightarrow Lösche 0000, 0100 und 0101

	0001	0110	0111	1000	1010	1100	1110	1111
--00				1		1		
-1-0		1				1	1	
-11-		1	1				1	1
1--0				1	1	1	1	
0-0-	1							
01--		1	1					

0110, 0111 und 1110 dominieren 1111 \Rightarrow Lösche 0110, 0111 und 1110

	0001	1000	1010	1100	1111
--00		1		1	
-1-0				1	
-11-					1
1--0		1	1	1	
0-0-	1				
01--					

1000 und 1100 dominieren 1010 \Rightarrow Lösche 1000 und 1100

	0001	1010	1111
--00			
-1-0			
-11-			1
1--0		1	
0-0-	1		
01--			

2te Regel nicht mehr anwendbar

- d) [1], Folgefehler sind ok

0-0-, 1--0 und -11- sind offensichtlich wesentlich.

Leere Tabelle, kein Petrick notwendig

e) [1], Folgefehler sind ok

Minimalpolynom: $\bar{x}_1\bar{x}_3 + x_1\bar{x}_4 + x_2x_3$ bzw. 0-0-+1--0+-11-.

f) [2],

- Kosten des Minimalpolynoms: $cost_1 = 3$, $cost_2 = 9$.

- Das vollständige Polynom: $\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3x_4 + \bar{x}_1x_2x_3\bar{x}_4 + \bar{x}_1x_2x_3x_4 + x_1\bar{x}_2\bar{x}_3\bar{x}_4 + x_1\bar{x}_2x_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 + x_1x_2x_3\bar{x}_4$

Kosten des vollständigen Polynoms: $cost_1 = 11$, $cost_2 = 48$.

Aufgabe 2 (4 Punkte)

Beim Zusammenfassen zweier Implikanten nach der Methode von Quine-McCluskey resultiert ebenfalls ein Implikant der gegebenen Funktion f .

Diese Aussage lässt sich für einen beliebigen Implikanten i , der x_j nicht enthält, durch folgende Boolesche Formel ausdrücken:

$$\left(((x_j \wedge i) \rightarrow f) \wedge ((\neg x_j \wedge i) \rightarrow f) \right) = (i \rightarrow f)$$

Führen Sie den Beweis dieser Formel ausschließlich mittels Umformungen nach den Gesetzen der Booleschen Algebra durch. Sie dürfen neben den Axiomen alle Folgerungen aus der Vorlesung benutzen.

Außerdem sei der Operator \rightarrow wie folgt definiert:

$$(a \rightarrow b) = (\neg a \vee b)$$

Geben Sie bei jeder Ihrer Umformungen an, welche Regeln/Axiome der Booleschen Algebra Sie verwendet haben.

Lösung:

Punkte: Regeln generell nicht angegeben [-2]; einzelne Regeln vergessen je nach Häufigkeit [-0.5] bis [-1.5] (Kommutativität muss nicht angegeben werden); Regel falsch angewandt oder falsche Regel verwendet [-0.5] – bei größeren Fehlern auch bis zu [-1.5]; Achtung! Viele verschiedene Lösungswege denkbar

$$\begin{aligned}
 & \left(((x_j \wedge i) \rightarrow f) \wedge ((\neg x_j \wedge i) \rightarrow f) \right) && \stackrel{\text{Afg}}{=} && \left((\neg(x_j \wedge i) \vee f) \wedge (\neg(\neg x_j \wedge i) \vee f) \right) \\
 & && \stackrel{\text{DeMorg}}{=} && \left(((\neg x_j \vee \neg i) \vee f) \wedge ((\neg \neg x_j \vee \neg i) \vee f) \right) \\
 & && \stackrel{\text{DoppKomp}}{=} && \left(((\neg x_j \vee \neg i) \vee f) \wedge ((x_j \vee \neg i) \vee f) \right) \\
 & && \stackrel{\text{Assoz}}{=} && \left((\neg x_j \vee (\neg i \vee f)) \wedge (x_j \vee (\neg i \vee f)) \right) \\
 & && \stackrel{\text{Dist}}{=} && \left((\neg x_j \wedge x_j) \vee (\neg i \vee f) \right) \\
 & && \stackrel{\text{Kompl}}{=} && (\neg i \vee f) \\
 & && \stackrel{\text{Afg}}{=} && (i \rightarrow f)
 \end{aligned}$$

Aufgabe 3 (5 + 4 Punkte)

Die Funktion $g : \mathbb{B}^4 \rightarrow \mathbb{B}$ sei gegeben durch die Menge ihrer Primimplikanten $Prim(g)$ mit

$$Prim(g) = \{\bar{x}_1x_2x_4, x_2\bar{x}_3x_4, x_1\bar{x}_3x_4, x_1\bar{x}_2x_4, \bar{x}_2x_3x_4, \bar{x}_1x_3x_4\}$$

- a) Bestimmen Sie *alle* Minimalpolynome von g mit Hilfe der Methode von Petrick. Sie können davon ausgehen, dass alle Primimplikanten die gleichen Kosten haben.
- b) Gibt es eine Boolesche Funktion mit einem Minimalpolynom, das nicht ausschließlich aus Primimplikanten besteht? Beweisen Sie Ihre Aussage.

Lösung:

a) **Punkte:**

korrekte PIT [1];
 Übertragung in Produktsumme korrekt [0,5];
 Ausmultiplizieren [1,5]:

Es dürfen Teile der Zwischenergebnisse weggelassen werden, wenn ersichtlich ist, warum (wurde in der Lösung auch gemacht); Fehler im Ausmultiplizieren [-0,5], Folgefehler sind ok;
 Pro korrekt bestimmten Minimalpolynom [1] (evt. Folgefehler beachten);

Die ON-Menge ist

$$\begin{aligned} ON(g) &= \{0011, 0101, 0111, 1001, 1011, 1101\} \\ &= \{3, 5, 7, 9, 11, 13\} \end{aligned}$$

Die PIs bezeichnen wir folgendermassen:

$$\begin{aligned} PI1 &= \bar{x}_1x_2x_4 \\ PI2 &= x_2\bar{x}_3x_4 \\ PI3 &= x_1\bar{x}_3x_4 \\ PI4 &= x_1\bar{x}_2x_4 \\ PI5 &= \bar{x}_2x_3x_4 \\ PI6 &= \bar{x}_1x_3x_4 \end{aligned}$$

Damit ergibt sich die PIT für Petricks Methode wie folgt:

PI	ON(g)	3	5	7	9	11	13
1		1	1				
2			1				1
3				1		1	
4					1	1	
5		1				1	
6		1		1			

Man erhält damit:

$$\begin{aligned}
&= (5+6)(1+2)(1+6)(3+4)(4+5)(2+3) \\
&= (15+25+16+26)(13+14+36+46)(24+34+25+35) \\
&= (135+145+1356+1456+ \\
&\quad 1235+1245+2356+2456+ \\
&\quad 136+146+136+146+ \\
&\quad 1236+1246+236+246)(24+34+25+35) \\
&\quad (\text{Kombinationen mit 4 PIs können ignoriert werden}) \\
&= (135+145+136+146+236+246)(24+34+25+35) \\
&= (12345+1345+1235+135+ \\
&\quad 1245+1345+1245+1345+ \\
&\quad 1236+1346+12356+1356+ \\
&\quad 1246+1346+12456+13456+ \\
&\quad 2346+2346+2356+2356+ \\
&\quad 246+2346+2456+23456) \\
&= 135+246
\end{aligned}$$

D.h. es gibt zwei Minimalpolynome:

$$\begin{aligned}
\text{MP}_1(g) &= \bar{x}_1x_2x_4 + x_1\bar{x}_3x_4 + \bar{x}_2x_3x_4 \\
\text{MP}_2(g) &= x_2\bar{x}_3x_4 + x_1\bar{x}_2x_4 + \bar{x}_1x_3x_4
\end{aligned}$$

- Nur “Nein”: [1]

Beweis durch Widerspruch: [4]

Angenommen, es gäbe eine solche Funktion f . Dann sei p das Minimalpolynom von f , welches nicht nur Primiplikanten enthält. Sei m ein Implikant von f , der in p enthalten ist und kein Primimplikant ist. Da m kein Primimplikant ist, gibt es einen Primimplikanten m' von f , der ein echtes Teilmomom von m ist, also aus m durch Streichen von Literalen hervorgeht. Da $\text{cost}(m') < \text{cost}(m)$, hat das Polynom p' , welches aus p durch Ersetzen von m durch m' hervorgeht, geringere Kosten als p . p ist also nicht minimal!

Abgabe: 6. Dezember 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 7. Dezember 2013

Technische Informatik Übungsblatt 6

Aufgabe 1 (3 + 4 + 3 Punkte)

- a) Berechnen Sie nach der Schulmethode im Binärsystem die Summe von
 - $\langle 1010101 \rangle$ und $\langle 0101011 \rangle$
 - $\langle 1011 \rangle$ und $\langle 0110 \rangle$.
- b) Zeichnen sie auf der Basis der Standardbibliothek $STD = \{and_2, or_2, exor_2, not\}$ einen Carry-Ripple Addierer für $n = 4$
- c) Simulieren Sie den Carry-Ripple-Addierer aus der Teilaufgabe b) mit den Eingaben $a = 1011$, $b = 0110$, und $c = 0$, und vergleichen Sie das Ergebnis mit Teilaufgabe a).

Aufgabe 2 (2 + 6 Punkte)

- a) Welche aus der Vorlesung bekannte Funktion wird durch den Schaltkreis im Abb. 1 realisiert?
- b) Geben Sie einen Schaltkreis für $exor_n(x_1, \dots, x_n) = (\sum_{i=1}^n x_i) \bmod 2$ über die Bibliothek $BIB_1 = \{nand_2\}$ an.
Der Schaltkreis soll möglichst geringe Kosten und möglichst geringe Tiefe haben. Geben Sie Kosten und Tiefe Ihrer Realisierung an. Vergleichen Sie mit Kosten und Tiefe des in der Vorlesung gezeigten Schaltkreises über $BIB_2 = \{exor_2\}$.

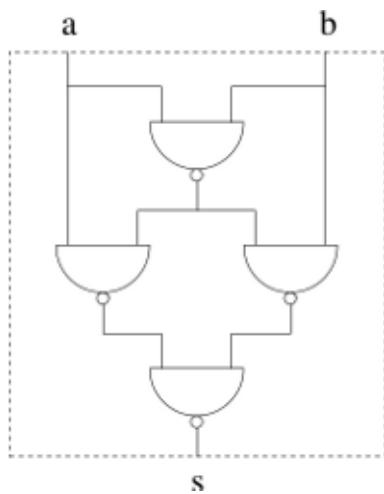


Abbildung 1: Schaltkreis

Aufgabe 3 ((0 + 6Bonus) + (0 + 6Bonus) Punkte)

- Wieviele verschiedene Pfade von einem beliebigen Eingang zum Ausgang gibt es im Schaltkreis aus Teilaufgabe 2.b) bzw. im Schaltkreis aus der Vorlesung zu exor_n ? Nehmen Sie dabei an, dass n eine Zweierpotenz ist.
- Geben Sie eine Realisierung zu exor_n über $BIB_1 = \{\text{nand}_2\}$ an, die exponentiell viele verschiedene Pfade von einem beliebigen Eingang zum Ausgang hat.

Abgabe: 14. Dezember 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
 Dr. Paolo Marin

Freiburg, 7. Dezember 2013

Technische Informatik Musterlösung zu Übungsblatt 6

Aufgabe 1 (3 + 4 + 3 Punkte)

- Berechnen Sie nach der Schulmethode im Binärsystem die Summe von
 - $\langle 1010101 \rangle$ und $\langle 0101011 \rangle$
 - $\langle 1011 \rangle$ und $\langle 0110 \rangle$.
- Zeichnen sie auf der Basis der Standardbibliothek $STD = \{and_2, or_2, exor_2, not\}$ einen Carry-Ripple Addierer für $n = 4$
- Simulieren Sie den Carry-Ripple-Addierer aus der Teilaufgabe b) mit den Eingaben $a = 1011$, $b = 0110$, und $c = 0$, und vergleichen Sie das Ergebnis mit Teilaufgabe a).

Lösung:

a)

- $\begin{array}{r} a < 1010101 > \\ +b < 0101011 > \\ \hline \end{array}$

1111111 Übertragung

- = 10000000
- $\begin{array}{r} a < 1011 > \\ +b < 0110 > \\ \hline \end{array}$

1110 Übertragung

- = 10001

- b) Hinweis: Die Notation "FA" anstatt "VA" verwenden.
 c) S. Abb 1.

Aufgabe 2 (2 + 6 Punkte)

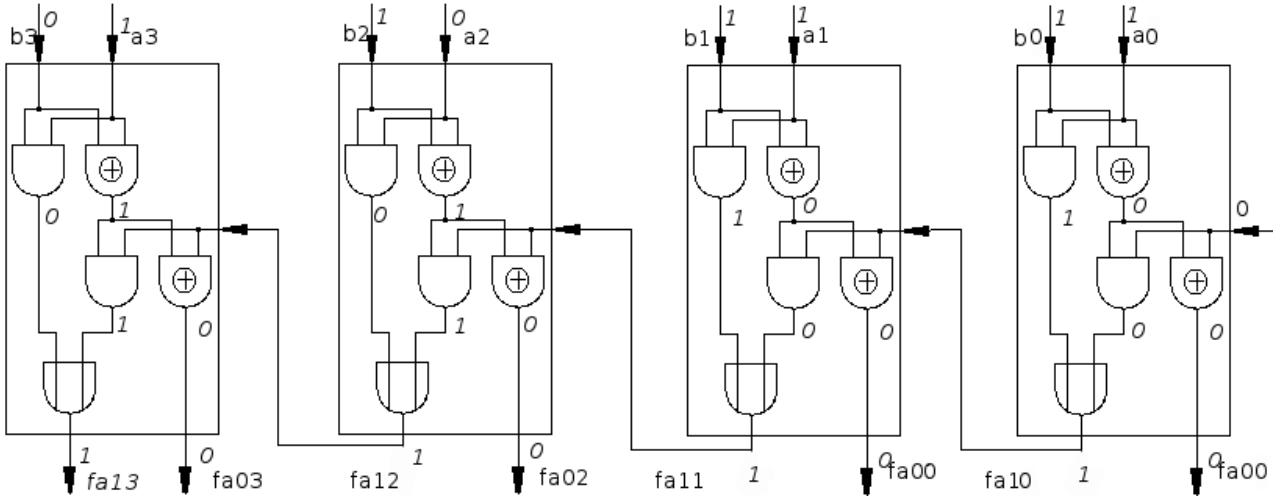


Abbildung 1: Simulation: Carry-Ripple-Addierer für $n = 4$ und Eingaben wie in der Aufgabe 1.a).

- a) Welche aus der Vorlesung bekannte Funktion wird durch den Schaltkreis im Abb. 2 realisiert?
- b) Geben Sie einen Schaltkreis für $\text{exor}_n(x_1, \dots, x_n) = (\sum_{i=1}^n x_i) \bmod 2$ über die Bibliothek $BIB_1 = \{\text{nand}_2\}$ an.
Der Schaltkreis soll möglichst geringe Kosten und möglichst geringe Tiefe haben. Geben Sie Kosten und Tiefe Ihrer Realisierung an. Vergleichen Sie mit Kosten und Tiefe des in der Vorlesung gezeigten Schaltkreises über $BIB_2 = \{\text{exor}_2\}$.

Lösung:

- a) (Hinweis: nur EXOR [-1])

$$s = \text{exor}_2(a, b)$$

Formal kann man zeigen:

$$\begin{aligned} s &= \text{nand}_2(\text{nand}_2(a, \text{nand}_2(a, b)), \text{nand}_2(\text{nand}_2(a, b), b)) \\ &\stackrel{\text{deMorgan}}{=} \text{or}_2(\text{and}_2(a, \text{nand}_2(a, b)), \text{and}_2(\text{nand}_2(a, b), b)) \\ &\stackrel{\text{Komm. u. Distr.}}{=} \text{and}_2(\text{nand}_2(a, b), \text{or}_2(a, b)) \\ &\stackrel{\text{deMorgan}}{=} \text{and}_2(\text{or}_2(\text{not}(a), \text{not}(b)), \text{or}_2(a, b)) \\ &\stackrel{\text{Def.}}{=} \text{exor}_2(a, b) \end{aligned}$$

Aber besser eine Funktionstabelle

durch Simulation erstellen.

a	b	N1	N2	N3	s
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

Tabelle 1: Funktionstabelle

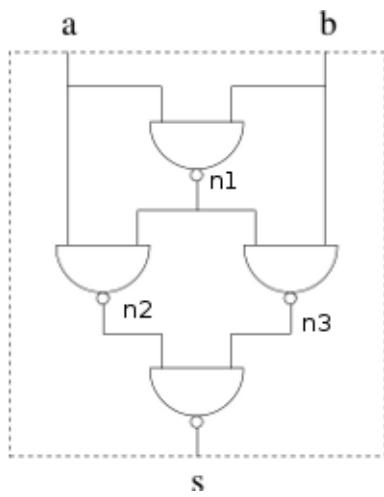
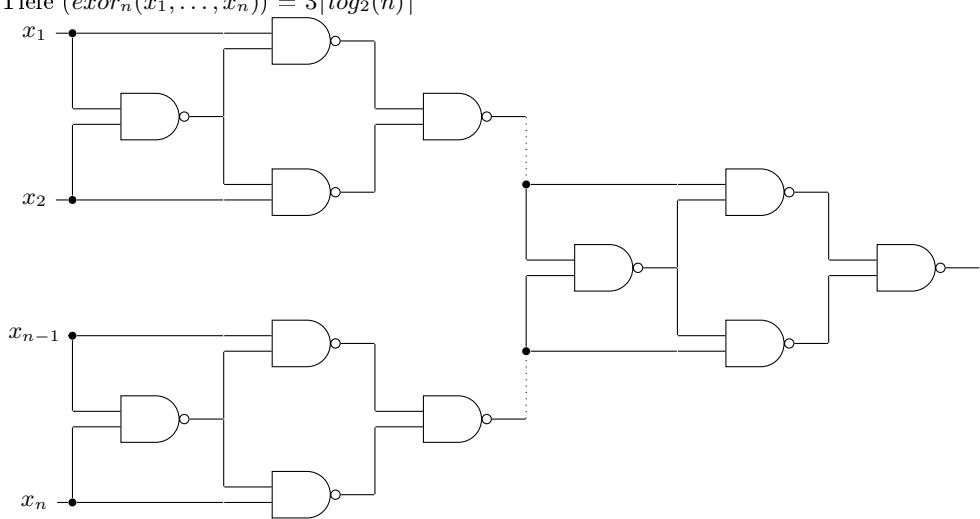


Abbildung 2: Schaltkreis

- b) Kosten ($\text{exor}_n(x_1, \dots, x_n)$) = $4(n - 1)$
 Tiefe ($\text{exor}_n(x_1, \dots, x_n)$) = $3\lceil \log_2(n) \rceil$



Aufgabe 3 ((0 + 6Bonus) + (0 + 6Bonus) Punkte)

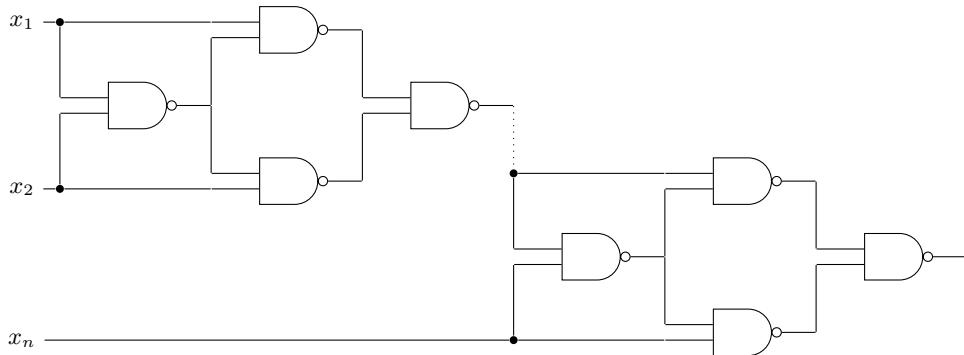
- a) Wieviele verschiedene Pfade von einem beliebigen Eingang zum Ausgang gibt es im Schaltkreis aus Teilaufgabe 2.b.) bzw. im Schaltkreis aus der Vorlesung zu $exor_n$? Nehmen Sie dabei an, dass n eine Zweierpotenz ist.

b) Geben Sie eine Realisierung zu $exor_n$ über $BIB_1 = \{nand_2\}$ an, die exponentiell viele verschiedene Pfade von einem beliebigen Eingang zum Ausgang hat.

Lösung:

- a) In der Vorlesung: n .
in der Aufgabe 2.b) $n \cdot 3^{\log_2 n} = n^{1+\log_2 3} \approx n^{2.58}$

b) Den "linearen" (oder wie Kaskade) Schaltkreis mit exor_2 -Gattern anzugeben, dann zu sagen: Jedes exor_2 -Gatter wird durch 4 nand_2 -Gatter ersetzt und dann erhält man 3^{n-1} Pfade von x_1 zum Ausgang.



Abgabe: 14. Dezember 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
 Dr. Paolo Marin

Freiburg, 13. Dezember 2013

Technische Informatik

Übungsblatt 7

Aufgabe 1 (4 + 3 Punkte)

- a) Zeichnen sie auf der Basis der Standardbibliothek $STD = \{and_2, or_2, exor_2, not\}$ einen Conditional Sum Addierer für $n = 4$
- b) Simulieren Sie den Conditional Sum Addierer aus der Teilaufgabe b) mit den Eingaben $a = 1011$, $b = 0110$, und $c = 0$.

Aufgabe 2 ((0 + 3 (Bonus)) + (0 + 3 (Bonus))) Punkte)

In der Vorlesung wurde den folgenden Satz vorgestellt (s. Folie 34, Kap. 3.5):

Seien $a, b \in \mathbb{B}^{n+1}$, $c_{-1} \in \{0, 1\}$ und $s \in \{0, 1\}^{n+1}$,

so dass $\langle c_n, s \rangle = \langle a \rangle + \langle b \rangle + c_{-1}$.

Dann gilt:

- $[a] + [b] + c_{-1} \notin R_n \Leftrightarrow (a_n = b_n) \wedge (b_n \neq s_n)$
- $[a] + [b] + c_{-1} \in R_n \Rightarrow [a] + [b] + c_{-1} = [s]$

Beweisen Sie den obigen Satz für den Fall $a_n = 0$ und $b_n = 1$ (oder umgekehrt).

Aufgabe 3 (2 + 3 Punkte)

Berechnen Sie das Ergebnis der folgenden Subtraktionen indem Sie die Subtraktion von Zweierkomplementzahlen auf geeignete Additionen zurückführen.

- a) $[1101] - [0011]$
- b) $[01010101] - [11110000]$

Aufgabe 4 (6 Punkte)

Geben Sie einen möglichst kleinen Schaltkreis an, der zu einer $(n + 1)$ -Bit-Zweierkomplementzahl deren Betrag und ggf. einen Überlauf berechnet, das heißt entwerfen Sie einen Schaltkreis zu der Booleschen Funktion

$$abs_{n+1} : \mathbb{B}^{n+1} \rightarrow \mathbb{B}^{n+2}, (a_n, \dots, a_0) \mapsto (ov, s_n, \dots, s_0)$$

mit $[s_n, \dots, s_0] = |[a_n, \dots, a_0]|$, $ov = 0$, falls $|[a_n, \dots, a_0]| \in \{-2^n, \dots, 2^n - 1\}$, $ov = 1$ sonst.
Bestimmen Sie die Kosten Ihrer Schaltkreisrealisierung.

Hinweis:

- Berücksichtigen Sie die Definition des Betrags einer Zahl r mit: $|r| = \begin{cases} r, & \text{falls } r \geq 0 \\ -r, & \text{falls } r < 0 \end{cases}$
Aus der Vorlesung bekannte Schaltkreise (wie Addierer, Inkrementer) dürfen Sie verwenden.
- Für die Überlaufbestimmung können Sie Resultate aus der Vorlesung verwenden. Für den Spezialfall in der Aufgabe können Sie sich aber auch überlegen, in welchen Fällen bzw. in welchem Fall der Betrag einer $(n + 1)$ -Bit-Zweierkomplementzahl nicht als $(n + 1)$ -Bit-Zweierkomplementzahl darstellbar ist.

Abgabe: 20. Dezember 2013, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 13. Dezember 2013

Technische Informatik Musterlösung zu Übungsblatt 7

Aufgabe 1 (4 + 3 Punkte)

- a) Zeichnen sie auf der Basis der Standardbibliothek $STD = \{and_2, or_2, exor_2, not\}$ einen Conditional Sum Addierer für $n = 4$
- b) Simulieren Sie den Conditional Sum Addierer aus der Teilaufgabe b) mit den Eingaben $a = 1011$, $b = 0110$, und $c = 0$.

Lösung:

- a) S. Abb 1.
- b) S. Abb 1.

Aufgabe 2 ((0 + 3 (Bonus)) + (0 + 3 (Bonus))) Punkte)

In der Vorlesung wurde den folgenden Satz vorgestellt (s. Folie 34, Kap. 3.5):

Seien $a, b \in \mathbb{B}^{n+1}$, $c_{-1} \in \{0, 1\}$ und $s \in \{0, 1\}^{n+1}$,

so dass $\langle c_n, s \rangle = \langle a \rangle + \langle b \rangle + c_{-1}$.

Dann gilt:

- $[a] + [b] + c_{-1} \notin R_n \Leftrightarrow (a_n = b_n) \wedge (b_n \neq s_n)$
- $[a] + [b] + c_{-1} \in R_n \Rightarrow [a] + [b] + c_{-1} = [s]$

Beweisen Sie den obigen Satz für den Fall $a_n = 0$ und $b_n = 1$ (oder umgekehrt).

Lösung:

(Aus der Vorlesung) Sei $a' = a_{n-1} \dots a_0$, $b' = b_{n-1} \dots b_0$, $s' = s_{n-1} \dots s_0$.

Dann gilt:

$$\begin{aligned} \langle a' \rangle + \langle b' \rangle + c_{-1} &= \langle c_{n-1} s' \rangle \\ c_{n-1} = 1 &\Leftrightarrow \langle a' \rangle + \langle b' \rangle + c_{-1} \geq 2^n \end{aligned}$$

$$[a] = \sum_{i=0}^{n-1} a_i \cdot 2^i - a_n \cdot 2^n = \langle a' \rangle - a_n \cdot 2^n$$

$$[b] = \langle b' \rangle - b_n \cdot 2^n.$$

Fall 3: $a_n = 0$ und $b_n = 1$ ($a_n = 1$ und $b_n = 0$ analog):

$$\begin{array}{ccccc} 0 & a_{n-1} & \dots & a_0 \\ 1 & b_{n-1} & \dots & b_0 \\ c_{n-1} & & & c_{-1} \\ \hline (s_{n+1}) & s_n & s_{n-1} & \dots & s_0 \end{array}$$

zu i):

a) Es gilt nach Fallannahme $a_n \neq b_n$

$$\begin{aligned} b) \quad & \underbrace{[a]}_{\langle a' \rangle} + \underbrace{[b]}_{\langle b' \rangle - 2^n} + c_{-1} = \underbrace{\langle a' \rangle}_{0 \leq \langle a' \rangle \leq 2^{n-1}} + \underbrace{\langle b' \rangle}_{0 \leq \langle b' \rangle \leq 2^{n-1}} - 2^n + c_{-1} \\ & -2^n \leq [a] + [b] + c_{-1} \leq 2^n - 1 \\ & \Rightarrow [a] + [b] + c_{-1} \in R_n \end{aligned}$$

zu ii):

Fall 3.1 $\langle a' \rangle + \langle b' \rangle + c_{-1} \geq 2^n \Leftrightarrow c_{n-1} = 1 \Rightarrow s_n = 0$

$$\begin{aligned} [a] + b] + c_{-1} &= \underbrace{\langle a' \rangle + \langle b' \rangle + c_{-1} - 2^n}_{=\langle c_{n-1} s' \rangle} \text{ da } a_n = 0, b_n = 1 \\ &= \langle c_{n-1} s' \rangle - 2^n \\ &= \langle s' \rangle + 2^n - 2^n \\ &= \langle s' \rangle \\ &= \underbrace{[0 s']}_{=s_n} \\ &= [s] \end{aligned}$$

Fall 3.2 $\langle a' \rangle + \langle b' \rangle + c_{-1} < 2^n \Leftrightarrow c_{n-1} = 0 \Rightarrow s_n = 1$

$$\begin{aligned} [a] + b] + c_{-1} &= \underbrace{\langle a' \rangle + \langle b' \rangle + c_{-1} - 2^n}_{=\langle c_{n-1} s' \rangle} \text{ (da } a_n = 0, b_n = 1) \\ &= \underbrace{\langle c_{n-1} s' \rangle}_{0} - 2^n \\ &= \langle s' \rangle - 2^n \\ &= \underbrace{[s_n s']}_{1} \\ &= [s] \end{aligned}$$

Aufgabe 3 (2 + 3 Punkte)

Berechnen Sie das Ergebnis der folgenden Subtraktionen indem Sie die Subtraktion von Zweierkomplementzahlen auf geeignete Additionen zurückführen.

- a) $[1101] - [0011]$
- b) $[01010101] - [11110000]$

Lösung:

Wege $-[b] = [\bar{b}] + 1$ kann $[a] - [b]$ zurückgeführt werden auf $[a] + [\bar{b}] + 1$

a) $[a] = [1101] = -3_{10}$, $[b] = [0011] = 3_{10}$, $[-b] = [1100]$

$$\begin{array}{r} 1101 \\ + 1100 \\ + \quad 1 \\ \hline (1)101 \quad (\text{übertragung}) \\ \hline \end{array}$$

$(1)1010 = -6_{10}$

b) $[a] = [01010101] = 85_{10}$, $[b] = [11110000] = -16_{10}$, $[-b] = [00001111]$

$$\begin{array}{r} 01010101 \\ + 00001111 \\ + \quad 1 \\ \hline (0)0011111 \quad (\text{übertragung}) \\ \hline \end{array}$$

$(0)01100101 = 101_{10}$

Aufgabe 4 (6 Punkte)

Geben Sie einen möglichst kleinen Schaltkreis an, der zu einer $(n + 1)$ -Bit-Zweierkomplementzahl deren Betrag und ggf. einen Überlauf berechnet, das heißt entwerfen Sie einen Schaltkreis zu der Booleschen Funktion

$$abs_{n+1} : \mathbb{B}^{n+1} \rightarrow \mathbb{B}^{n+2}, (a_n, \dots, a_0) \mapsto (ov, s_n, \dots, s_0)$$

mit $[s_n, \dots, s_0] = |[a_n, \dots, a_0]|$, $ov = 0$, falls $|[a_n, \dots, a_0]| \in \{-2^n, \dots, 2^n - 1\}$, $ov = 1$ sonst.
Bestimmen Sie die Kosten Ihrer Schaltkreisrealisierung.

Hinweis:

- Berücksichtigen Sie die Definition des Betrags einer Zahl r mit: $|r| = \begin{cases} r, & \text{falls } r \geq 0 \\ -r, & \text{falls } r < 0 \end{cases}$
Aus der Vorlesung bekannte Schaltkreise (wie Addierer, Inkrementer) dürfen Sie verwenden.
- Für die Überlaufbestimmung können Sie Resultate aus der Vorlesung verwenden. Für den Spezialfall in der Aufgabe können Sie sich aber auch überlegen, in welchen Fällen bzw. in welchem Fall der Betrag einer $(n + 1)$ -Bit-Zweierkomplementzahl nicht als $(n + 1)$ -Bit-Zweierkomplementzahl darstellbar ist.

Lösung:

a) S. Abb 2.

b) Kosten:

$$C(SK) = C(INC_n) + n \cdot C(XOR-Reihe) = n \cdot C(HA) + n = 3n$$

Abgabe: 20. Dezember 2013, 17⁰⁰ über das Übungsportal

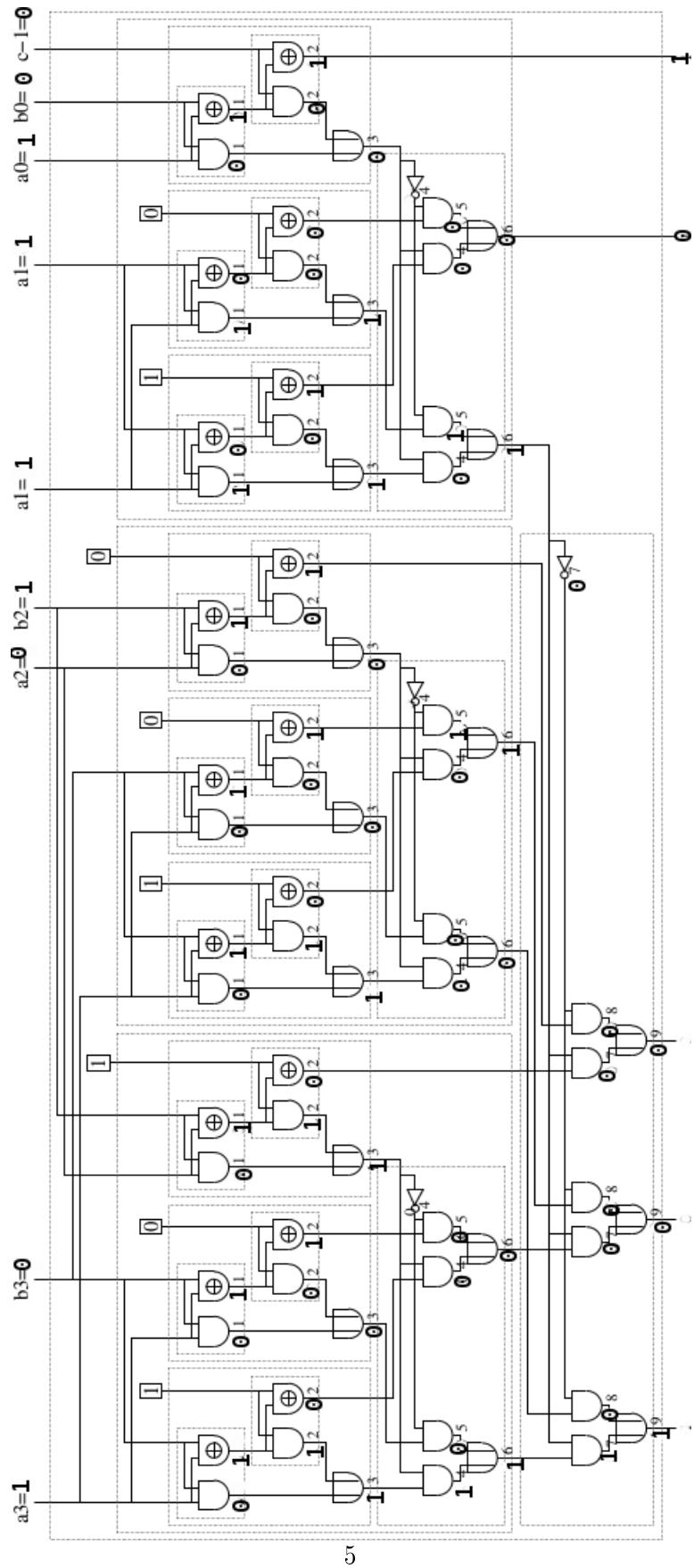


Abbildung 1: Simulation: Conditional Sum Addierer für $n = 4$.

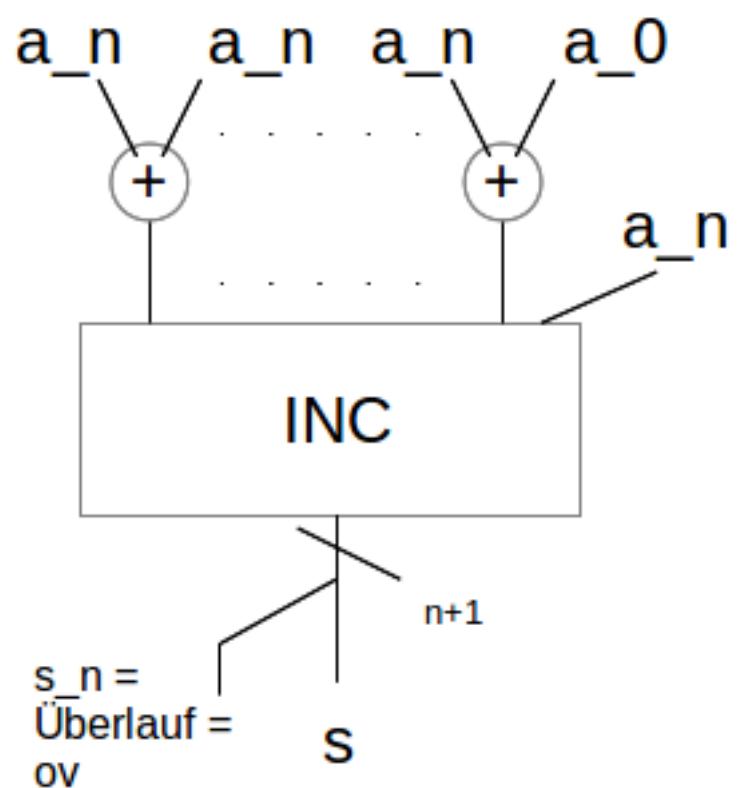


Abbildung 2: Schaltkreis

Prof. Dr. Christoph Scholl
 Dr. Paolo Marin

Freiburg, 20. Dezember 2013

Technische Informatik

Übungsblatt 8

Aufgabe 1 (4 + 2 Punkte)

- a) In der Vorlesung wurde für die Tiefe des Carry-Ripple-Addierers CR_n und des Conditional-Sum-Addierers CSA_n gezeigt:

$$\text{depth}(CR_n) = 2(n - 1) + 3$$

$$\text{depth}(CSA_n) = 3\log n + 3$$

Betrachten Sie Addierer, deren Bitbreite $n = 2^k$ eine Zweierpotenz ist.

- Berechnen Sie $\text{depth}(CR_n)$ und $\text{depth}(CSA_n)$ für $k = 0, 1, \dots, 16$ und fassen Sie die Ergebnisse in einer Tabelle zusammen.
- Für welche Bitbreite hat der Conditional-Sum-Addierer eine größere Tiefe als der Carry-Ripple-Addierer?

- b) Zeigen Sie:

i) $4\log^2 n + 10\log n + 6 \in O(\log^2 n)$

ii) $5n + 3\log n - 7 \in O(n)$

- c) Zeigen Sie die folgende Aussage, die für den Spezialfall $k = l = 1$ in der Vorlesung benutzt wurde:

Falls $f \in O(n^k)$ und $g \in O(n^l)$ für $l, k \in \mathbb{N}$, dann ist $f \cdot g \in O(n^{k+l})$.

Aufgabe 2 (2 + 3 Punkte)

Berechnen Sie nach der Schulmethode das Ergebnis der folgenden Multiplikationen. Kennzeichnen Sie die Partialprodukte PP_i .

- a) $[1101] \times [0011]$
 b) $[01010101] \times [11110000]$

Aufgabe 3 (4 + 4 Punkte)

Seien $d^0, \dots, d^{2^m-1} \in \{0, 1\}^n$, $s_0, \dots, s_{m-1} \in \{0, 1\}$. Ein verallgemeinerter Multiplexer berechnet eine Boolesche Funktion $\text{alg_mux}_{n,m} : \{0, 1\}^{2^m \cdot n + m} \mapsto \{0, 1\}^n$, $\text{alg_mux}(d^0, \dots, d^{2^m-1}, s_0, \dots, s_{m-1}) = d^i$, falls $\langle s_{m-1} \dots s_0 \rangle = i$.

- Benutzen Sie 3 n-Bit-Multiplexer MUX_n , um einen verallgemeinerten Multiplexer zu $\text{alg_mux}_{n,2}$ zu konstruieren.
- Geben Sie eine rekursive Darstellung für $\text{alg_mux}_{n,m}$ an. Geben Sie dazu wie üblich eine Realisierung für den Basisfall $m = 1$ an und eine Realisierung, die einen verallgemeinerten Multiplexer für $m \in N$ unter Zuhilfenahme von verallgemeinerten Multiplexern für $m - 1$ konstruiert. n-Bit-Multiplexer MUX_n dürfen Sie in Ihrer Realisierung verwenden. Berechnen Sie Kosten und Tiefe Ihrer Schaltkreisrealisierung für $\text{alg_mux}_{n,m}$.

Aufgabe 4 ((0 + 3 (Bonus)) Punkte)

Zeigen Sie formal, dass bei der Addition von drei Binärzahlen u, v und w zu zwei Teilsummen s und c mit einem n Bit Carry-Save-Addierer folgende Gleichung gilt.

$$\langle u \rangle + \langle v \rangle + \langle w \rangle = \langle s \rangle + \langle c \rangle$$

Hinweis: Sie dürfen voraussetzen, dass für einen Volladdierer FA gilt:

$$FA(x, y, z) = (fa_1, fa_0) \text{ mit } \langle fa_1, fa_0 \rangle = \langle x \rangle + \langle y \rangle + \langle z \rangle$$

Aufgabe 5 (6 Punkte)

Ein Multiplizierer für zwei n-Bit Binärzahlen (siehe Abb. 1), wie er in der Vorlesung vorgestellt wurde, berechnet die Funktion

$$mul_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n} \text{ mit}$$

$$mul_n(a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0) = (p_{2n-1}, \dots, p_0) \text{ und } \langle p_{2n-1}, \dots, p_0 \rangle = \langle a \rangle \cdot \langle b \rangle.$$

Geben Sie aufbauend auf dieser Definition einen Schaltkreis an, der zwei n-Bit *Zweierkomplementzahlen* multiplizieren kann.

$$twoc_mul_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n} \text{ mit}$$

$$twoc_mul_n(a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0) = (p_{2n-1}, \dots, p_0) \text{ und } [p_{2n-1}, \dots, p_0]_2 = [a]_2 \cdot [b]_2.$$

Erklären Sie zusätzlich kurz die Idee Ihrer Lösung.

Hinweis: Sie dürfen für Ihre Lösung die arithmetischen Schaltungen, die in der Vorlesung vorgestellt wurden, verwenden (z.B. Addierer, Inkrementer, Multiplizierer,...). Für diese Teilschaltkreise ist es ausreichend, in der Schaltung einen entsprechend beschriftetes Rechteck einzuleichen.

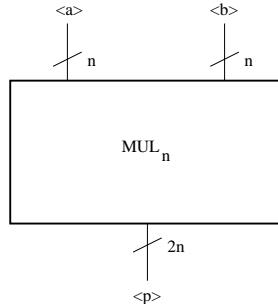


Abbildung 1: n -Bit Multiplizierer

Aufgabe 6 (0 Punkte)

Der *iCub* (www.iCub.org) ist ein seit 2004 entwickelter humanoider Roboter, der heute einer der bekanntesten Roboter weltweit ist. Teile des Prozessors wurden im *Italian Institute of Technology* (www.iit.it) konstruiert. Dafür war es notwendig neue Logik-Gatter zu entwerfen, die die humanoiden Funktionen optimal umsetzen. In Abb. 2 sind einige Prototypen dieser neuen Logik-Gatter Familie dargestellt.

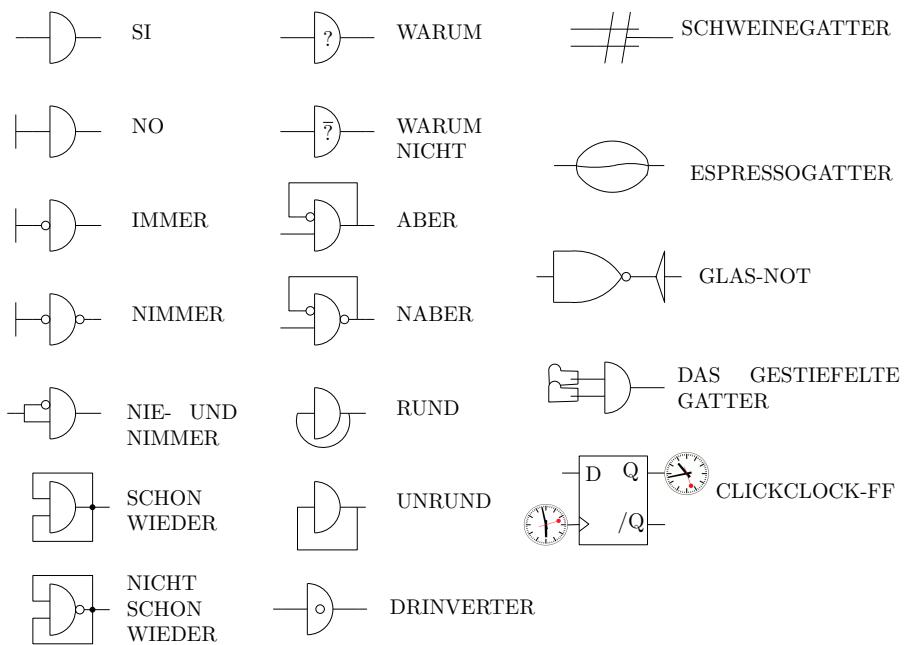


Abbildung 2: Logikgatter der iCub-Technologie

Entwerfen Sie weitere Gattertypen bzw. Gattersymbole für die Familie der iCub-Technologie in diesem Stil. Falls Sie es für notwendig halten, geben Sie eine entsprechende Erklärung der Funktionsweise des Gatters an. Seien Sie kreativ!

Abgabe: 10. Januar 2014, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 20. Dezember 2013

Technische Informatik Musterlösung zu Übungsblatt 8

Aufgabe 1 (4 + 2 Punkte)

- a) In der Vorlesung wurde für die Tiefe des Carry-Ripple-Addierers CR_n und des Conditional-Sum-Addierers CSA_n gezeigt:

$$\text{depth}(CR_n) = 2(n - 1) + 3$$

$$\text{depth}(CSA_n) = 3\log n + 3$$

Betrachten Sie Addierer, deren Bitbreite $n = 2^k$ eine Zweierpotenz ist.

- Berechnen Sie $\text{depth}(CR_n)$ und $\text{depth}(CSA_n)$ für $k = 0, 1, \dots, 16$ und fassen Sie die Ergebnisse in einer Tabelle zusammen.
- Für welche Bitbreite hat der Conditional-Sum-Addierer eine größere Tiefe als der Carry-Ripple-Addierer?

- b) Zeigen Sie:

i) $4\log^2 n + 10\log n + 6 \in O(\log^2 n)$

ii) $5n + 3\log n - 7 \in O(n)$

- c) Zeigen Sie die folgende Aussage, die für den Spezialfall $k = l = 1$ in der Vorlesung benutzt wurde:

Falls $f \in O(n^k)$ und $g \in O(n^l)$ für $l, k \in \mathbb{N}$, dann ist $f \cdot g \in O(n^{k+l})$.

Lösung:

a)	•	k	n	$depth(CR_n)$	$depth(CSA_n)$
		0	1	3	3
		1	2	5	6
		2	4	9	9
		3	8	17	12
		4	16	33	15
		5	32	65	18
		6	64	129	21
		7	128	257	24
		8	256	513	27
		9	512	1025	30
		10	1024	2049	33
		11	2048	4097	36
		12	4096	8193	39
		13	8192	16385	42
		14	16384	32769	45
		15	32768	65537	48
		16	65536	131073	51

- Tiefe des Conditional-Sum-Addierers $depth(CSA_n) = 3\log n + 3$ größer als die Tiefe des Carry-Ripple-Addierers $depth(CRA_n) = 2(n - 1) + 3$:

Aus der Tabelle sieht man, dass die einzige Bitbreite wodurch $depth(CSA_n) > depth(CRA_n)$ gilt ist $n = 2$.

- b) i) $4\log^2 n + 10\log n + 6 \in O(\log^2 n) \Leftrightarrow \exists c, x_0 \in \mathbb{R}_0^+$ mit $4\log^2 n + 10\log n + 6 \leq c \cdot \log^2 n$ f.a. $n > x_0$
 $c = 5, x_0 = 2^{11}$

Beweis:

$$\begin{aligned} 4\log^2 n + 10\log n + 6 &\leq 4\log^2 n + 10\log n + \log n = 4\log^2 n + 11\log n \leq 5\log^2 n \\ 11\log n &\leq \log^2 n \\ 11 &\leq \log n \\ n &\geq 2^{11} \end{aligned}$$

- i) $5n + 3\log n - 7 \in O(n) \Leftrightarrow \exists c, x_0 \in \mathbb{R}_0^+$ mit $5n + 3\log n - 7 \leq c \cdot n$ f.a. $n > x_0$
 $c = 8$

Beweis:

$$\begin{aligned} 5n + 3\log n - 7 &< 5n + 3n - 7 = 8n - 7 \leq 8n \\ -7 &\leq 0 \end{aligned}$$

- c) Es gibt $c_0, x_0 \in \mathbb{R}_0^+$ so dass $f \leq c_0 \cdot n^k$ für alle $n > x_0$ gilt, $c_1, x_1 \in \mathbb{R}_0^+$ so dass $g \leq c_1 \cdot n^l$ für alle $n > x_1$ gilt, dann $f \in O(n^k)$ und $g \in O(n^l)$, dann ist $f \cdot g \in O(n^{k+l})$.

Es gibt $c_2, x_2 \in \mathbb{R}_0^+$ mit $c_2 = c_0 \cdot c_1$ so dass $f \cdot g \leq c_0 n^k \cdot c_1 n^l \leq c_2 \cdot n^{k+l}$ für alle $n > x_2 \geq \max(x_0, x_1)$ gilt.

Aufgabe 2 (2 + 3 Punkte)

Berechnen Sie nach der Schulmethode das Ergebnis der folgenden Multiplikationen. Kennzeichnen Sie die Partialprodukte PP_i .

- a) $[1101] \times [0011]$
b) $[01010101] \times [11110000]$

Lösung:

a) $[1101] \times [0011]$

$$\begin{array}{r}
 1101 \quad pp0 \\
 + 1101 \quad pp1 \\
 + 0000 \quad pp2 \\
 + 0000 \quad pp3 \\
 \hline
 0100111
 \end{array}$$

b) $[01010101] \times [11110000]$

$$\begin{array}{r}
 00000000 \quad pp0 \\
 + 00000000 \quad pp1 \\
 + 00000000 \quad pp2 \\
 + 00000000 \quad pp3 \\
 + 01010101 \quad pp4 \\
 + 01010101 \quad pp5 \\
 + 01010101 \quad pp6 \\
 + 01010101 \quad pp7 \\
 \hline
 100111110110000
 \end{array}$$

Aufgabe 3 (4 + 4 Punkte)

Seien $d^0, \dots, d^{2^m-1} \in \{0, 1\}^n$, $s_0, \dots, s_{m-1} \in \{0, 1\}$. Ein verallgemeinerter Multiplexer berechnet eine Boolesche Funktion $alg_mux_{n,m} : \{0, 1\}^{2^m \cdot n + m} \mapsto \{0, 1\}^n$, $alg_mux(d^0, \dots, d^{2^m-1}, s_0, \dots, s_{m-1}) = d^i$, falls $\langle s_{m-1} \dots s_0 \rangle = i$.

- a) Benutzen Sie 3 n-Bit-Multiplexer MUX_n , um einen verallgemeinerten Multiplexer zu $alg_mux_{n,2}$ zu konstruieren.
- b) Geben Sie eine rekursive Darstellung für $alg_mux_{n,m}$ an. Geben Sie dazu wie üblich eine Realisierung für den Basisfall $m = 1$ an und eine Realisierung, die einen verallgemeinerten Multiplexer für $m \in N$ unter Zuhilfenahme von verallgemeinerten Multiplexern für $m - 1$ konstruiert. n-Bit-Multiplexer MUX_n dürfen Sie in Ihrer Realisierung verwenden. Berechnen Sie Kosten und Tiefe Ihrer Schaltkreisrealisierung für $alg_mux_{n,m}$.

Lösung:

- a) Siehe Abb. 1.

- b) Basisfall $m = 1$: standard MUX_n .
Dann für eine allgemeine m : (Siehe Abb. 2)

$$\begin{aligned}
 \text{Kosten} &= (2^m - 1) \cdot K(MUX_n) = (2^m - 1) \cdot (3n + 1) \\
 \text{Tiefe} &= m \cdot T(MUX_n) = m \cdot 3
 \end{aligned}$$

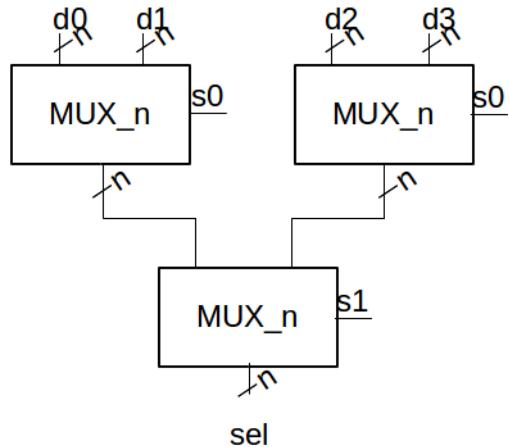


Abbildung 1: Realisierung von $alg_mux_{n,2}$

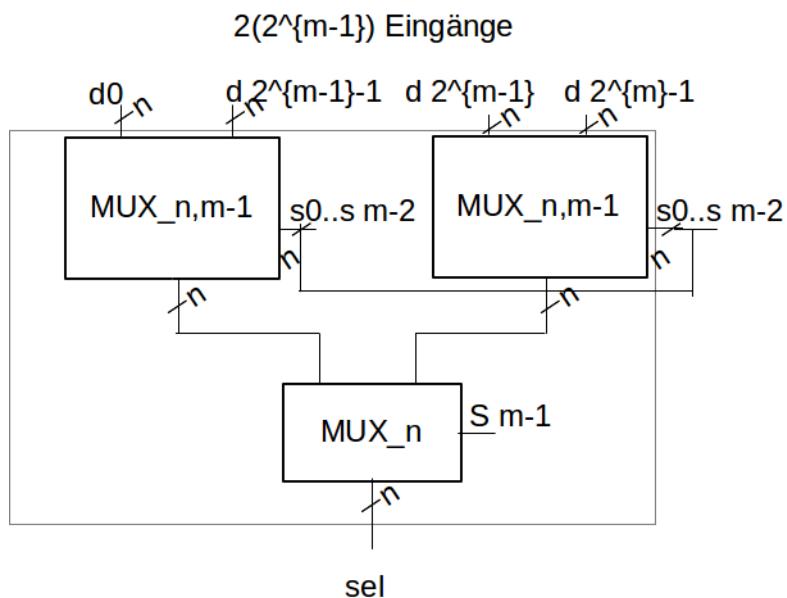


Abbildung 2: Rekursive Konstruktion von $alg_mux_{n,m}$

Aufgabe 4 ((0 + 3 (Bonus)) Punkte)

Zeigen Sie formal, dass bei der Addition von drei Binärzahlen u , v und w zu zwei Teilsummen s und c mit einem n Bit Carry-Save-Addierer folgende Gleichung gilt.

$$\langle u \rangle + \langle v \rangle + \langle w \rangle = \langle s \rangle + \langle c \rangle$$

Hinweis: Sie dürfen voraussetzen, dass für einen Volladdierer FA gilt:

$$FA(x, y, z) = (\text{fa}_1, \text{fa}_0) \text{ mit } \langle \text{fa}_1, \text{fa}_0 \rangle = \langle x \rangle + \langle y \rangle + \langle z \rangle$$

Lösung:

Da der CSavA aus einzelnen Volladdieren und einem konstanten Ausgang $c_0 = 0$ besteht, gilt:

$$\langle s \rangle + \langle c \rangle = \sum_{i=0}^{n-1} s_i \cdot 2^i + \sum_{i=0}^n c_i \cdot 2^i \quad (1)$$

$$= \sum_{i=0}^{n-1} (2c_{i+1} + s_i) \cdot 2^i, \text{ da } c_0 = s_n = 0 \quad (2)$$

$$= \sum_{i=0}^{n-1} \langle c_{i+1}, s_i \rangle \cdot 2^i \quad (3)$$

$$= \sum_{i=0}^{n-1} (u_i + v_i + w_i) \cdot 2^i, \text{ wegen } FA \quad (4)$$

$$= \langle u \rangle + \langle v \rangle + \langle w \rangle. \quad (5)$$

Punkte: Ansatz [1]; Definition von FA verwendet [1]

Aufgabe 5 (6 Punkte)

Ein Multiplizierer für zwei n -Bit Binärzahlen (siehe Abb. 3), wie er in der Vorlesung vorgestellt wurde, berechnet die Funktion

$$mul_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n} \text{ mit}$$

$$mul_n(a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0) = (p_{2n-1}, \dots, p_0) \text{ und } \langle p_{2n-1}, \dots, p_0 \rangle = \langle a \rangle \cdot \langle b \rangle.$$

Geben Sie aufbauend auf dieser Definition einen Schaltkreis an, der zwei n -Bit *Zweierkomplementzahlen* multiplizieren kann.

$$twoc_mul_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n} \text{ mit}$$

$$twoc_mul_n(a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0) = (p_{2n-1}, \dots, p_0) \text{ und } [p_{2n-1}, \dots, p_0]_2 = [a]_2 \cdot [b]_2.$$

Erklären Sie zusätzlich kurz die Idee Ihrer Lösung.

Hinweis: Sie dürfen für Ihre Lösung die arithmetischen Schaltungen, die in der Vorlesung vorgestellt wurden, verwenden (z.B. Addierer, Inkrementer, Multiplizierer,...). Für diese Teilschaltkreise ist es ausreichend, in der Schaltung einen entsprechend beschriftetes Rechteck einzuleichen.

Lösung:

Gundidee: Wenn eine der Zahlen negativ ist, bestimme den Betrag (leistet Subtrahierer). Korrigiere nach der Multiplikation das Ergebnis wieder entsprechend. Bei positiven Zahlen ist keine Korrektur notwendig.

In Abbildung 4 ist eine Lösungsmöglichkeit dargestellt. Hier wird ein Subtrahierer zur betragsbildung benutzt.

Ein alternativer Vorschlag zur Bildung des Betrags ist in Abb. 5 dargestellt. Wenn Eingang $i = 1$ gilt wird das Negat gebildet. Ein Überlauf kann nicht vorkommen. Bei $i = 0$ wird mit 0 inkrementiert und die Zahl wird nicht

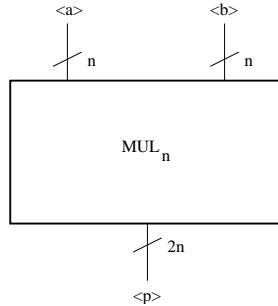


Abbildung 3: n -Bit Multiplizierer

verändert. Bei Betragsbildung $i = a_{n-1}$ gesetzt wird. (Bzw. bei Berechnung der negativen Zahl in Abb. 4 unten eben $i = a_{n-1} \otimes b_{n-1}$.)

Aufgabe 6 (0 Punkte)

Der *iCub* (www.iCub.org) ist ein seit 2004 entwickelter humanoider Roboter, der heute einer der bekanntesten Roboter weltweit ist. Teile des Prozessors wurden im *Italian Institute of Technology* (www.iit.it) konstruiert. Dafür war es notwendig neue Logik-Gatter zu entwerfen, die die humanoiden Funktionen optimal umsetzen. In Abb. 6 sind einige Prototypen dieser neuen Logik-Gatter Familie dargestellt.

Entwerfen Sie weitere Gattertypen bzw. Gattersymbole für die Familie der iCub-Technologie in diesem Stil. Falls Sie es für notwendig halten, geben Sie eine entsprechende Erklärung der Funktionsweise des Gatters an. Seien Sie kreativ!

Lösung:

Keine Musterlösung.

Punkte: Keine. Ein Best-Of bitte an mich schicken.

Abgabe: 10. Januar 2014, 17⁰⁰ über das Übungsportal

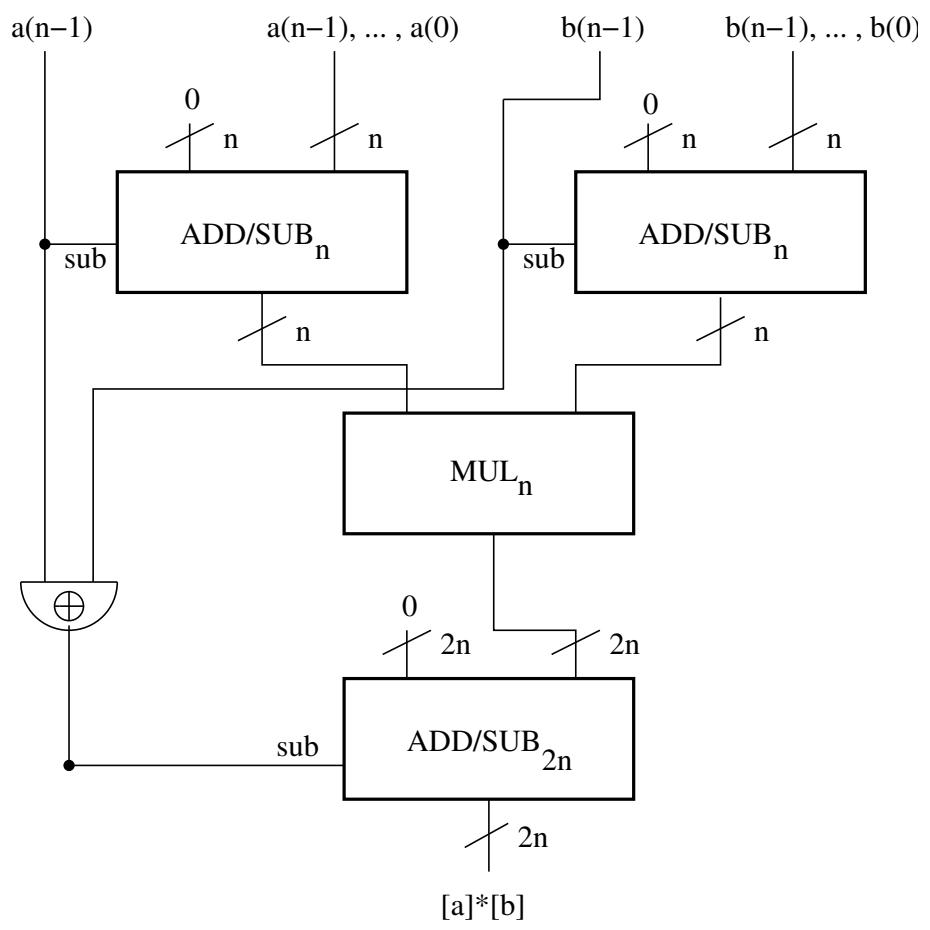


Abbildung 4: Multiplizierer für Zweierkomplementzahlen

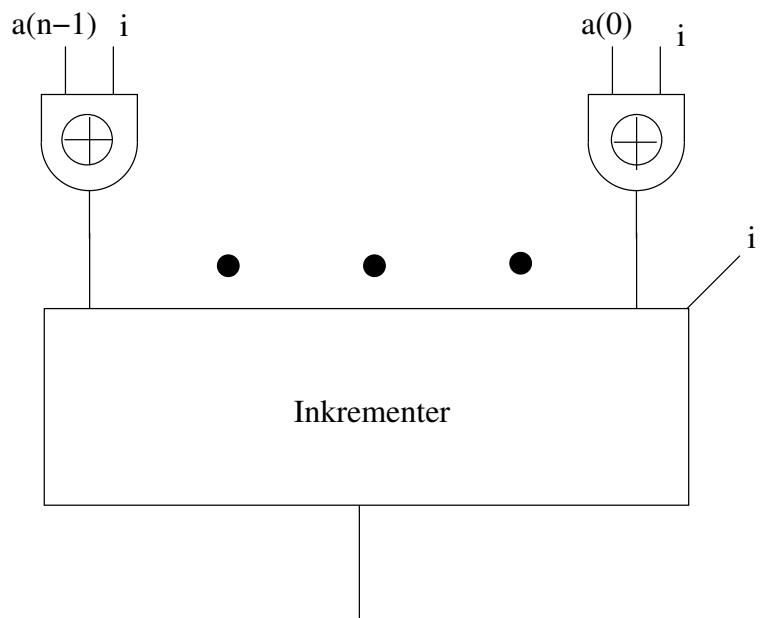


Abbildung 5: Bedingte Negation einer 2er-Komplementzahl

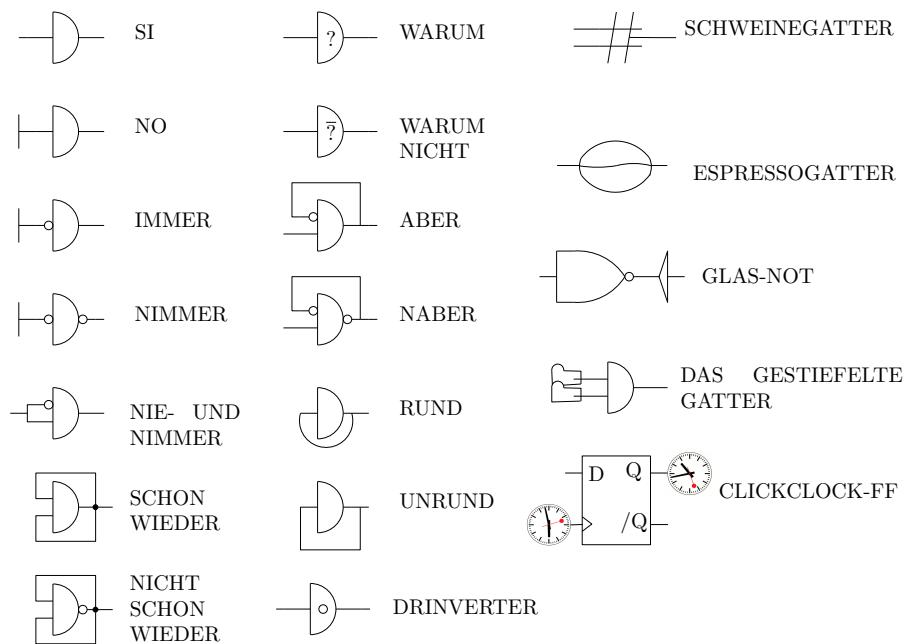


Abbildung 6: Logikgatter der iCub-Technologie

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 10. Januar 2014

Technische Informatik

Übungsblatt 8

Aufgabe 1 (2 + 2 + 1 + 2 Punkte)

Ihnen gefällt das in der Vorlesung vorgestellte RS-Flipflop nicht, weswegen Sie beschließen, sich einen anderen, aber relativ ähnlichen Schaltkreis aus OR- und NOT Bausteinen anzusehen:

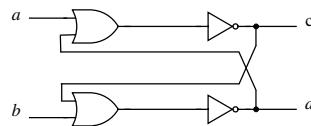


Abbildung 1: Kein RS-Flipflop

- Welche stabile Belegungen gibt es für diesen Schaltkreis? Geben Sie dazu die Werte für a , b , c und d an.
- Geben Sie an, bei welcher Eingangsbelegung ein gespeicherter Wert gehalten wird und durch welche Eingangsbelegungen ein neuer Wert gespeichert werden kann.
- Sind a und b active-low oder active-high?
- Welche der stabilen Belegungen macht bei der Verwendung als speicherndes Element keinen Sinn? Was ist der Grund dafür?

Aufgabe 2 (3 + 4 Punkte)

In Abb. 2 ist ein Mealy-Automat mit zwei Eingangsvariablen a , b , vier Zuständen s_{00} , s_{01} , s_{10} , s_{11} und einem Ausgang y angegeben. Beachten Sie, dass anstelle von einzelnen Belegungen der Eingangsvariablen a und b in der Abbildung Boolesche Ausdrücke über a und b angegeben sind. Wenn e eine Kante ist, die von Zustand s zu Zustand t führt, wenn e mit einem Booleschen Ausdruck f und einem Ausgabesymbol δ beschriftet ist und f die erfüllenden Belegungen $\epsilon_1, \dots, \epsilon_k$ hat, dann ist diese Kante eine abkürzende Schreibweise für k Kanten von s nach t , die jeweils mit ϵ_i / δ beschriftet sind. Die Zustände $s_{ij} \in S$ können durch zwei Zustandsvariablen z_1 und z_0 kodiert werden. Dabei entspricht die Zustandskodierung dem Index der Zustände; z_1 gibt den Wert des Bits i , z_0 den des Bits j wieder.

Beachten Sie, dass gilt: $a, b, y \in \mathbb{B}$.

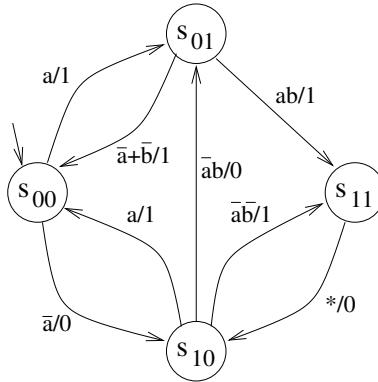


Abbildung 2: Mealy-Automat ('*' bedeutet a, b beliebig)

- Geben Sie die Zustandsübergangstabelle an.
- Geben Sie für z_1^{t+1} und z_0^{t+1} jeweils die Übergangsfunktion in DNF an. Spezifizieren Sie ebenso die Ausgabefunktion für den Ausgang y .

Aufgabe 3 (3 Punkte)

Definition Wir erweitern die Definition eines Mealy-Automaten aus der Vorlesung um Endzustände in folgender Weise: Ein *Mealy-Automat mit Endzuständen* ist ein 7-Tupel $M = (S, \Sigma, \Delta, \delta, \lambda, S_0, S_e)$ mit:

S	ist eine endliche Menge von Zuständen
Σ	ist ein Eingabealphabet
Δ	ist ein Ausgabealphabet
$\delta : S \times \Sigma \rightarrow S$	ist die Übergangsfunktion
$\lambda : S \times \Sigma \rightarrow \Delta$	ist die Ausgabefunktion
$S_0 \subseteq S$	ist die Menge der Startzustände
$S_e \subseteq S$	ist die Menge der Endzustände

Der Mealy-Automat M startet in einem Zustand $s_0 \in S_0$ und liest eine Eingabe $w = w_1 w_2 \dots w_n \in \Sigma^*$ Zeichen für Zeichen. Nach jedem gelesenen Zeichen wechselt der Mealy-Automat M abhängig von dem aktuellen Zustand und dem gerade gelesenen Zeichen in den durch δ bestimmten neuen Zustand und gibt das durch λ bestimmte Zeichen aus.

Beispiel

$$M_1 = (S, \Sigma, \Delta, \delta, \lambda, S_0, S_e)$$

mit: $S = \{s_0, s_1\}$

$$\Sigma = \{0, 1\}$$

$$\Delta = \{a, b, c\}$$

$$S_0 = \{s_0\}$$

$$S_e = \{s_1\}$$

Die Übergangsfunktion und die Ausgabefunktion kann in einer *Zustandsübergangstabelle* angegeben werden:

$s \in S$	$\sigma \in \Sigma$	$\delta(s, \sigma)$	$\lambda(s, \sigma)$
s_0	0	s_0	a
s_0	1	s_1	b
s_1	0	s_1	a
s_1	1	s_0	c

Ein Mealy-Automat kann auch als *Zustandsdiagramm* angegeben werden (siehe Abbildung 3).

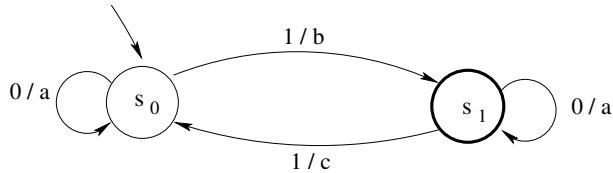


Abbildung 3: Zustandsdiagramm von M_1

An den Pfeilen steht jeweils die Eingabe, bei der dieser Übergang ausgeführt wird, und die dabei erfolgte Ausgabe. Die Startzustände werden durch einen zusätzlichen eingehenden Pfeil gekennzeichnet. Die Endzustände sind fett markiert.

Betrachten Sie den Mealy-Automaten M_1 . Gültige Eingabeworte sind all diejenigen, nach deren Abarbeitung von einem Startzustand aus sich der Automat in einem Endzustand befindet.

Gegeben seien die folgenden Eingabeworte:

$$w_1 = 00010$$

$$w_2 = 11001$$

$$w_3 = 101101011$$

Welche Zustände durchläuft M_1 für die Eingabeworte w_1, w_2, w_3 und welche dieser Eingabeworte sind gültig? Wie lautet die jeweilige Ausgabesequenz?

Aufgabe 4 (4 + 3 + (0 + 2 (Bonus)) Punkte)

Betrachten Sie den Mealy-Automaten in Abb. 4. Der Startzustand ist mit einem eingehenden Pfeil gekennzeichnet (s_0).

- Überführen Sie den Automaten in einen äquivalenten Moore-Automaten.
- Geben Sie eine allgemeine Vorgehensweise an, um einen Mealy-Automaten in einen Moore-Automaten zu überführen. Geben Sie außerdem eine allgemeine Vorgehensweise an, um einen Moore-Automaten in einen Mealy-Automaten zu überführen.
- Asymptotisch wie viele Zustände hat ein aus einem Mealy-Automaten entstandener Moore-Automat im Vergleich zum ausgehenden Mealy-Automaten mit Ihrem in c) angegebenen Verfahren?

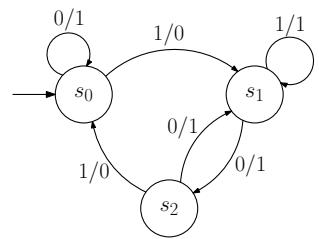


Abbildung 4: Mealy-Automat

Abgabe: 17. Januar 2014, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 10. Januar 2014

Technische Informatik

Musterlösung zu Übungsblatt 8

Aufgabe 1 (2 + 2 + 1 + 2 Punkte)

Ihnen gefällt das in der Vorlesung vorgestellte RS-Flipflop nicht, weswegen Sie beschließen, sich einen anderen, aber relativ ähnlichen Schaltkreis aus OR- und NOT Bausteinen anzusehen:

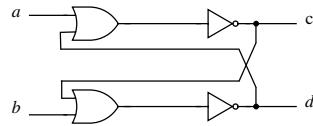


Abbildung 1: Kein RS-Flipflop

- Welche stabile Belegungen gibt es für diesen Schaltkreis? Geben Sie dazu die Werte für a , b , c und d an.
- Geben Sie an, bei welcher Eingangsbelegung ein gespeicherter Wert gehalten wird und durch welche Eingangsbelegungen ein neuer Wert gespeichert werden kann.
- Sind a und b active-low oder active-high?
- Welche der stabilen Belegungen macht bei der Verwendung als speicherndes Element keinen Sinn? Was ist der Grund dafür?

Lösung:

- Es gibt fünf stabile Belegungen:
 - $a = 0, b = 0, c = 0, d = 1$
 - $a = 0, b = 0, c = 1, d = 0$
 - $a = 0, b = 1, c = 1, d = 0$
 - $a = 1, b = 0, c = 0, d = 1$
 - $a = 1, b = 1, c = 0, d = 0$

- b) Bei $a = b = 0$ wird der aktuelle Wert gehalten, mit $a = 0, b = 1$ wird c auf 1 und d auf 0 gesetzt und mit $a = 1, b = 0$ wird c auf 0 und d auf 1 gesetzt.
- c) a und b sind active-high, da sie durch das Heben auf 1 aktiviert werden.
- d) Die Belegung $a = 1, b = 1$ macht keinen Sinn, da es bei gleichzeitigem Absenken von a und b zu Flackern kommen kann.

Aufgabe 2 (3 + 4 Punkte)

In Abb. 2 ist ein Mealy-Automat mit zwei Eingangsvariablen a, b , vier Zuständen $s_{00}, s_{01}, s_{10}, s_{11}$ und einem Ausgang y angegeben. Beachten Sie, dass anstelle von einzelnen Belegungen der Eingangsvariablen a und b in der Abbildung Boolesche Ausdrücke über a und b angegeben sind. Wenn e eine Kante ist, die von Zustand s zu Zustand t führt, wenn e mit einem Booleschen Ausdruck f und einem Ausgabesymbol δ beschriftet ist und f die erfüllenden Belegungen $\epsilon_1, \dots, \epsilon_k$ hat, dann ist diese Kante eine abkürzende Schreibweise für k Kanten von s nach t , die jeweils mit ϵ_i / δ beschriftet sind. Die Zustände $s_{ij} \in S$ können durch zwei Zustandsvariablen z_1 und z_0 kodiert werden. Dabei entspricht die Zustandskodierung dem Index der Zustände; z_1 gibt den Wert des Bits i , z_0 den des Bits j wieder.

Beachten Sie, dass gilt: $a, b, y \in \mathbb{B}$.

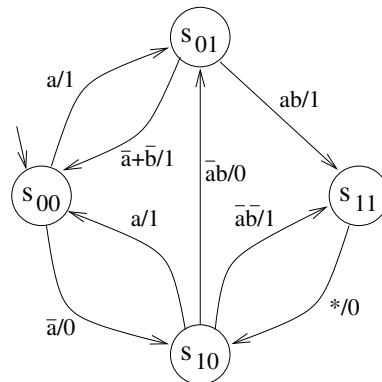


Abbildung 2: Mealy-Automat ('*' bedeutet a, b beliebig)

- a) Geben Sie die Zustandsübergangstabelle an.
- b) Geben Sie für z_1^{t+1} und z_0^{t+1} jeweils die Übergangsfunktion in DNF an. Spezifizieren Sie ebenso die Ausgabefunktion für den Ausgang y .

Lösung:

zu a:

a	b	z_1^t	z_0^t	z_1^{t+1}	z_0^{t+1}	y
1	-	0	0	0	1	1
0	-	0	0	1	0	0
1	1	0	1	1	1	1
0	-	0	1	0	0	1
-	0	0	1	0	0	1
1	-	1	0	0	0	1
0	1	1	0	0	1	0
0	0	1	0	1	1	1
-	-	1	1	1	0	0

zu b:

z_1^{t+1}	$z_1^t z_0^t$	$z_1^t \bar{z}_0^t$	$\bar{z}_1^t z_0^t$	$\bar{z}_1^t \bar{z}_0^t$
ab	1	0	0	1
$a\bar{b}$	1	0	0	0
$\bar{a}b$	1	1	1	0
$\bar{a}\bar{b}$	1	0	1	0

$$z_1^{t+1} = z_1^t z_0^t + \bar{z}_1^t \bar{z}_0^t \bar{a} + z_0^t ab + z_1^t \bar{a} \bar{b}$$

z_0^{t+1}	$z_1^t z_0^t$	$z_1^t \bar{z}_0^t$	$\bar{z}_1^t z_0^t$	$\bar{z}_1^t \bar{z}_0^t$
ab	0	0	1	1
$a\bar{b}$	0	0	1	0
$\bar{a}b$	0	1	0	0
$\bar{a}\bar{b}$	0	1	0	0

$$z_0^{t+1} = z_1^t \bar{z}_0^t \bar{a} + \bar{z}_1^t \bar{z}_0^t a + \bar{z}_1^t ab$$

y	$z_1^t z_0^t$	$z_1^t \bar{z}_0^t$	$\bar{z}_1^t z_0^t$	$\bar{z}_1^t \bar{z}_0^t$
ab	0	1	1	1
$a\bar{b}$	0	1	1	1
$\bar{a}b$	0	1	0	1
$\bar{a}\bar{b}$	0	0	0	1

$$y = \bar{z}_0^t a + \bar{z}_1^t z_0^t + z_1^t \bar{z}_0^t \bar{b}$$

Aufgabe 3 (3 Punkte)

Definition Wir erweitern die Definition eines Mealy-Automaten aus der Vorlesung um Endzustände in folgender Weise: Ein *Mealy-Automat mit Endzuständen* ist ein 7-Tupel $M = (S, \Sigma, \Delta, \delta, \lambda, S_0, S_e)$ mit:

- S ist eine endliche Menge von Zuständen
- Σ ist ein Eingabealphabet
- Δ ist ein Ausgabealphabet
- $\delta : S \times \Sigma \rightarrow S$ ist die Übergangsfunktion
- $\lambda : S \times \Sigma \rightarrow \Delta$ ist die Ausgabefunktion
- $S_0 \subseteq S$ ist die Menge der Startzustände
- $S_e \subseteq S$ ist die Menge der der Endzustände

Der Mealy-Automat M startet in einem Zustand $s_0 \in S_0$ und liest eine Eingabe $w = w_1 w_2 \dots w_n \in \Sigma^*$ Zeichen für Zeichen. Nach jedem gelesenen Zeichen wechselt der Mealy-Automat M abhängig

von dem aktuellen Zustand und dem gerade gelesenen Zeichen in den durch δ bestimmten neuen Zustand und gibt das durch λ bestimmte Zeichen aus.

Beispiel

$$M_1 = (S, \Sigma, \Delta, \delta, \lambda, S_0, S_e)$$

$$\text{mit: } S = \{s_0, s_1\}$$

$$\Sigma = \{0, 1\}$$

$$\Delta = \{a, b, c\}$$

$$S_0 = \{s_0\}$$

$$S_e = \{s_1\}$$

Die Übergangsfunktion und die Ausgabefunktion kann in einer *Zustandsübergangstabelle* angegeben werden:

$s \in S$	$\sigma \in \Sigma$	$\delta(s, \sigma)$	$\lambda(s, \sigma)$
s_0	0	s_0	a
s_0	1	s_1	b
s_1	0	s_1	a
s_1	1	s_0	c

Ein Mealy-Automat kann auch als *Zustandsdiagramm* angegeben werden (siehe Abbildung 3).

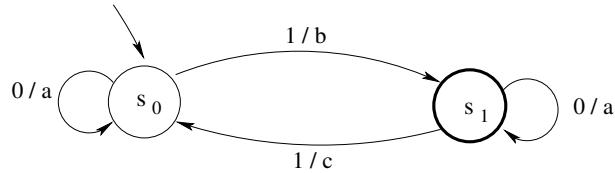


Abbildung 3: Zustandsdiagramm von M_1

An den Pfeilen steht jeweils die Eingabe, bei der dieser Übergang ausgeführt wird, und die dabei erfolgte Ausgabe. Die Startzustände werden durch einen zusätzlichen eingehenden Pfeil gekennzeichnet. Die Endzustände sind fett markiert.

Betrachten Sie den Mealy-Automaten M_1 . Gültige Eingabeworte sind all diejenigen, nach deren Abarbeitung von einem Startzustand aus sich der Automat in einem Endzustand befindet.

Gegeben seien die folgenden Eingabeworte:

$$w_1 = 00010$$

$$w_2 = 11001$$

$$w_3 = 101101011$$

Welche Zustände durchläuft M_1 für die Eingabeworte w_1, w_2, w_3 und welche dieser Eingabeworte sind gültig? Wie lautet die jeweilige Ausgabesequenz?

Lösung:

$$w_1: s_0 \rightarrow s_0 \rightarrow s_0 \rightarrow s_0 \rightarrow s_1 \rightarrow s_1; \quad \text{gültig.}$$

Ausgabe: aaaba

$$w_2: s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_0 \rightarrow s_0 \rightarrow s_1; \quad \text{gültig.}$$

Ausgabe: bcaab

$$w_3: s_0 \rightarrow s_1 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_1 \rightarrow s_0 \rightarrow s_0 \rightarrow s_1 \rightarrow s_0; \quad \text{nicht gültig.}$$

Ausgabe: bacbacacb

Aufgabe 4 (4 + 3 + (0 + 2 (Bonus)) Punkte)

Betrachten Sie den Mealy-Automaten in Abb. 4. Der Startzustand ist mit einem eingehenden Pfeil gekennzeichnet (s_0).

- Überführen Sie den Automaten in einen äquivalenten Moore-Automaten.
- Geben Sie eine allgemeine Vorgehensweise an, um einen Mealy-Automaten in einen Moore-Automaten zu überführen. Geben Sie außerdem eine allgemeine Vorgehensweise an, um einen Moore-Automaten in einen Mealy-Automaten zu überführen.
- Asymptotisch wie viele Zustände hat ein aus einem Mealy-Automaten entstandener Moore-Automat im Vergleich zum ausgehenden Mealy-Automaten mit Ihrem in c) angegebenen Verfahren?

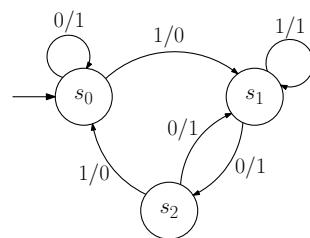


Abbildung 4: Mealy-Automat

Lösung:

- s. Abbildung 5; Punkte [-1] wenn λ nicht vom aktuellen, sondern vom nächsten Zustand abhängt.

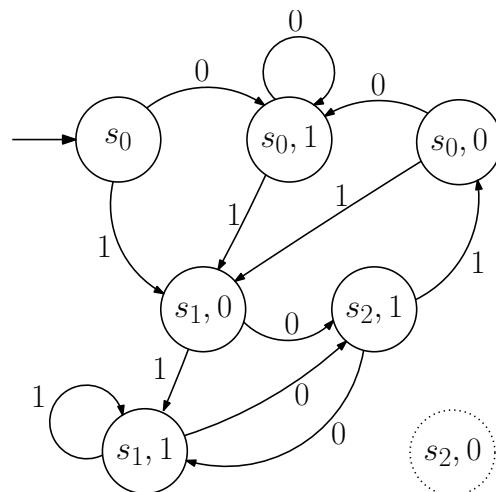


Abbildung 5: Moore-Automat

b) Moore \rightarrow Mealy: [1]

- Beschrifte alle eingehenden Kanten eines Zustandes mit der Ausgabe des Zustandes.

Mealy \rightarrow Moore: [2] Keine gesonderte Betrachtung der Initialzustände notwendig.

- Erstelle für jede Kombination von Zuständen und Eingabe einen neuen Zustand ($S^{\text{neu}} = \{(s, o) | s \in S, o \in O\}$);
- Erstelle für jeden Startzustand einen weiteren Zustand (S^{init}). Dies sind die Startzustände des Moore-Automaten.
- Für jeden Übergang aus S^{neu} mit der Beschriftung i/o zwischen zwei Zuständen s_j und s_k des Mealy-Automaten, füge im Moore-Automaten einen Übergang von s_j, m zu s_k, o mit der Kantenbeschriftung i ein (für alle $m \in O$). Für jede ausgehende Kante mit der Beschriftung i/o eines Startzustandes des Mealy-Automaten füge zusätzlich eine Kante mit der Beschriftung i von s nach s_k, o ein (für alle $s \in S^{\text{init}}$). Siehe Abb. 6

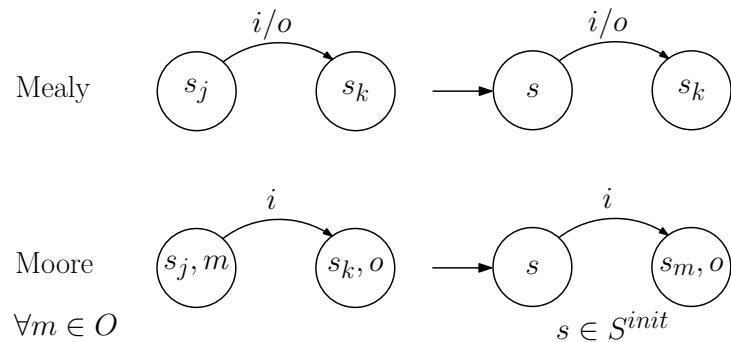


Abbildung 6: Illustration des Vorgehens.

c) $\mathbf{O}(|O| \times |S| + |S_0|) = \mathbf{O}(|O| \times |S|)$

Abgabe: 17. Januar 2014, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 17. Januar 2014

Technische Informatik

Übungsblatt 10

Aufgabe 1 (3 + 3 + 3 + 3 Punkte)

Betrachten Sie den allgemeinen Aufbau eines SRAMs, so wie er in der Vorlesung vorgestellt wurde (Kap. 4.4, Folie 5).

- Zeichnen Sie zunächst einen 2-Bit-Dekodierer D_2 nur mit den Grundgattern (Standardbibliothek BIB = {and; or; xor; not}). Die Treiber können dabei vernachlässigt werden, da der Ausgangsgrad kleiner als zehn ist.
- Zeichnen Sie nun ein 4-Bit SRAM. Dabei sollen der Dekodierer und das mehrfach-OR in Grundgatter aufgelöst werden. Latches dürfen verwendet werden, Treiberbäume können wiederum vernachlässigt werden.
- Entwerfen Sie einen 4-Bit-Dekodierer, indem Sie auf einem 2-Bit-Dekodierer aufbauen. Zeichnen Sie die Schaltung.
- Gegeben sei eine Funktionstabelle mit dem Inputvektor A (3-Bit Adresse) sowie dem Outputvektor Y (4-Bit Binärzahl), wobei Y sich aus der Binärcodierung Ihres persönlichen Geburtsdatums ergibt, wie in der folgenden Beispieltabelle für das Datum 14.05.1970 demonstriert:

A₂	A₁	A₀	Y₃	Y₂	Y₁	Y₀	Zahl
0	0	0	0	0	0	1	1
0	0	1	0	1	0	0	4
0	1	0	0	0	0	0	0
0	1	1	0	1	0	1	5
1	0	0	0	0	0	1	1
1	0	1	1	0	0	1	9
1	1	0	0	1	1	1	7
1	1	1	0	0	0	0	0

Entwerfen Sie für Ihre ganz persönliche Funktion einen Schaltkreis über der Bibliothek, die ausschließlich Dekodierer und mehrfache OR's (wie in der Vorlesung vorgestellt) enthält. Zeichnen und begründen Sie Ihr Ergebnis.

Aufgabe 2 (2 + (0 + 4 (Bonus)) + 2 Punkte)

Zeigen Sie, dass für jedes $b, s \in \mathbb{N}$ und $x \in \{1, \dots, b^s\}$ ein Baum $T(x, s)$ mit Ausgangsgrad $\leq b$ und den folgenden Eigenschaften existiert:

- $T(x, s)$ hat x Blätter.
- Für die Zahl der inneren Knoten gilt: $I(T(x, s)) \leq \frac{x}{b-1} + s$.
- Alle Pfade von der Wurzel zu einem Blatt in $T(x, s)$ haben Länge s .

Hinweis: Vollständige Induktion über s .

Zur Erinnerung: In der Vorlesung wurde angedeutet, dass man Treiberbäume auf einen solchen Baum $T(x, s)$ zurückführen kann. Die Anzahl der inneren Knoten von T entspricht hierbei den Kosten und die Pfadlänge von T der Tiefe des Treiberbaumes. Aus der Tiefe lässt sich dann die Verzögerungszeit bestimmen.

Aufgabe 3 (4 + 4 Punkte)

Betrachten Sie einen neuen Rechner mit den Registern PC (Program Counter), ACC (Akkumulator), B (Operandenregister) und IN (Indexregister), wie in Abbildung 1 skizziert. Die Instruktionen des Rechners sind in Tabelle 3 zusammengestellt. Dabei werden die Register wieder durch 2 Bits kodiert, die in den Befehlen mit D und S bezeichnet werden. Die Operationscodes op und die Bedingungen c sind entsprechend des RETI Rechners gewählt.

- Zeigen Sie, dass die Instruktionen des neuen Rechners mit den Datenpfaden aus Abbildung 1 durchgeführt werden können.
- Können alle $LOAD$, $STORE$ und $COMPUTE$ -Befehle des RETI Rechners auf dem neuen Rechner simuliert werden? Geben Sie entsprechende Befehlsfolgen und die notwendigen Einschränkungen für den neuen Rechner an.

I[31:30]	I[29:28]	I[27]	I[26]	I[25:0]	Befehl	Wirkung
LOAD-Befehle: $D \in \{PC, IN, B, ACC\}$ $i \in \mathbb{B}^{26}$						
01	00	D	i	$LOAD D i$	$D := M(i)$	$PC := PC + 1$
01	01	D	*	$LOADIN D$	$D := M(IN)$	$PC := PC + 1$
01	10	D	i	$LOADI D i$	$D := i$	$PC := PC + 1$
STORE-Befehle: $S \in \{PC, IN\}$ $i \in \mathbb{B}^{26}$						
10	00	*	*	$STORE i$	$M(i) := ACC$	$PC := PC + 1$
10	10	*	*	$STOREIN$	$M(IN) := ACC$	$PC := PC + 1$
10	01	S	*	$MOVE S$	$ACC := S$	$PC := PC + 1$
COMPUTE-Befehle: $op \in \{SUB, ADD, OPLUS, OR, AND\}$						
00	op	*	*	op	$ACC := ACC op B$	$PC := PC + 1$

Tabelle 1: Instruktionen des neuen Rechners

Abgabe: 24. Januar 2014, 17⁰⁰ über das Übungsportal

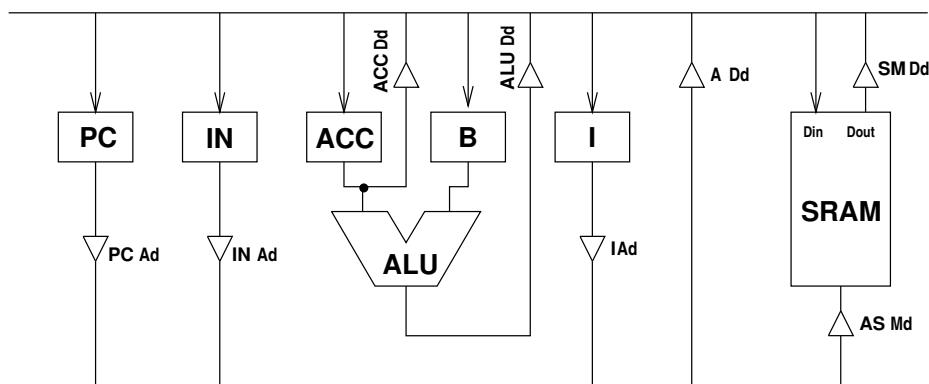


Abbildung 1: Datenpfade des neuen Rechners

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 17. Januar 2014

Technische Informatik

Musterlösung zu Übungsblatt 10

Aufgabe 1 (3 + 3 + 3 + 3 Punkte)

Betrachten Sie den allgemeinen Aufbau eines SRAMs, so wie er in der Vorlesung vorgestellt wurde (Kap. 4.4, Folie 5).

- Zeichnen Sie zunächst einen 2-Bit-Dekodierer D_2 nur mit den Grundgattern (Standardbibliothek BIB = {and; or; xor; not}). Die Treiber können dabei vernachlässigt werden, da der Ausgangsgrad kleiner als zehn ist.
- Zeichnen Sie nun ein 4-Bit SRAM. Dabei sollen der Dekodierer und das mehrfach-OR in Grundgatter aufgelöst werden. Latches dürfen verwendet werden, Treiberbäume können wiederum vernachlässigt werden.
- Entwerfen Sie einen 4-Bit-Dekodierer, indem Sie auf einem 2-Bit-Dekodierer aufbauen. Zeichnen Sie die Schaltung.
- Gegeben sei eine Funktionstabelle mit dem Inputvektor A (3-Bit Adresse) sowie dem Outputvektor Y (4-Bit Binärzahl), wobei Y sich aus der Binärcodierung Ihres persönliches Geburtsdatums ergibt, wie in der folgenden Beispieltabelle für das Datum 14.05.1970 demonstriert:

A₂	A₁	A₀	Y₃	Y₂	Y₁	Y₀	Zahl
0	0	0	0	0	0	1	1
0	0	1	0	1	0	0	4
0	1	0	0	0	0	0	0
0	1	1	0	1	0	1	5
1	0	0	0	0	0	1	1
1	0	1	1	0	0	1	9
1	1	0	0	1	1	1	7
1	1	1	0	0	0	0	0

Entwerfen Sie für Ihre ganz persönliche Funktion einen Schaltkreis über der Bibliothek, die ausschließlich Dekodierer und mehrfache OR's (wie in der Vorlesung vorgestellt) enthält. Zeichnen und begründen Sie Ihr Ergebnis.

Lösung:

- a) siehe Abbildung 1

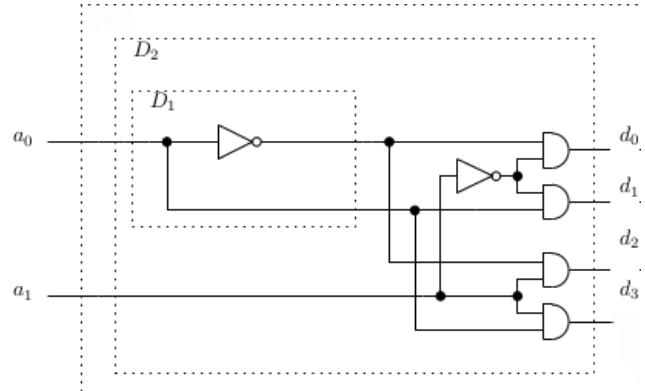


Abbildung 1: 3-Bit-Dekodierer

- b) siehe Abbildung 2

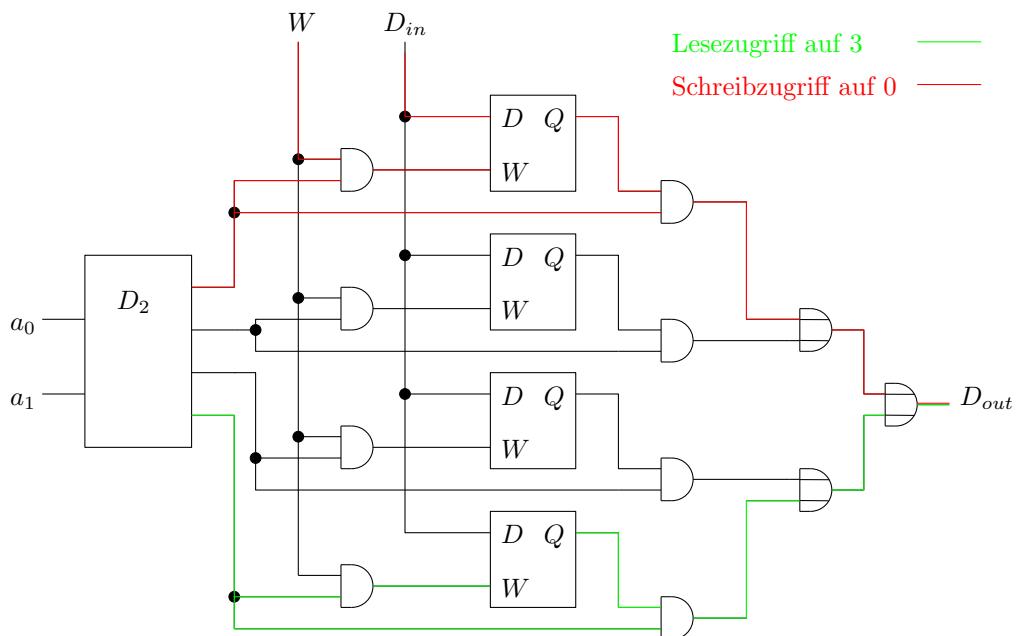


Abbildung 2: 4-Bit SRAM

- c) siehe Abbildung 3
- d) siehe Abbildung 4 Im Prinzip ist das doch der Teil eines 8x4-Bit-SRAM, der für Lesen zuständig ist und bei dem die D-Latches durch Konstanten ersetzt sind, d.h. im Prinzip ein ROM (Read Only Memory). Nach Konstantenpropagation fallen alle Und-Gatter an den D-Latches und im Fall von 0-Konstanten auch die entsprechenden Eingänge der Oder-Gatter weg.

Punkte: bei a) und b) irgendwie: richtig oder falsch, Kleinere Fehler (Gatter vergessen, falsche Gatter, fehlende Verbindungen) [-0,5]

Aufgabe 2 (2 + (0 + 4 (Bonus)) + 2 Punkte)

Zeigen Sie, dass für jedes $b, s \in \mathbb{N}$ und $x \in \{1, \dots, b^s\}$ ein Baum $T(x, s)$ mit Ausgangsgrad $\leq b$ und den folgenden Eigenschaften existiert:

- $T(x, s)$ hat x Blätter.
- Für die Zahl der inneren Knoten gilt: $I(T(x, s)) \leq \frac{x}{b-1} + s$.
- Alle Pfade von der Wurzel zu einem Blatt in $T(x, s)$ haben Länge s .

Hinweis: Vollständige Induktion über s .

Zur Erinnerung: In der Vorlesung wurde angedeutet, dass man Treiberbäume auf einen solchen Baum $T(x, s)$ zurückführen kann. Die Anzahl der inneren Knoten von T entspricht hierbei den Kosten und die Pfadlänge von T der Tiefe des Treiberbaumes. Aus der Tiefe lässt sich dann die Verzögerungszeit bestimmen.

Lösung:

Induktion über s .

$s = 1$:

- $x \in \{1, \dots, b\}$.
- Konstruiere $T(x, 1)$ wie in Abbildung 5
- $T(x, 1)$ hat x Blätter.
- Jeder Pfad hat Länge $s = 1$.
- $I(T(x, 1)) = 1 \leq \frac{x}{b-1} + 1$.

$(s - 1) \rightarrow s$:

- Sei $x = a \cdot b^{s-1} + d$ mit $a \in \{0, \dots, b\}$, $d \in \{0, \dots, b^{s-1}\}$.
- Konstruiere $T(x, s)$ nach Induktionsvoraussetzung mit $A = T(b^{s-1}, s-1)$ und $D = T(d, s-1)$ wie in Abbildung 6
- Weil a und d nicht beide 0 sein können, existiert zumindest einer der beiden Typen A oder D .
- Nach Induktionsvoraussetzung hat jeder Teilbaum die Pfadlänge $s - 1$ und somit hat $T(x, s)$ die Pfadlänge s .
- Nach Induktionsvoraussetzung hat jeder Teilbaum b^{s-1} beziehungsweise d Blätter. $T(x, s)$ hat folglich $x = a \cdot b^{s-1} + d$ Blätter.
- Zur Abschätzung der Zahl der inneren Knoten müssen wir für die vollständigen Binärbäume $T(b^{s-1}, s-1)$ die Zahl der inneren Knoten exakt ausrechnen. Diese ist

$$I(T(b^{s-1}, s-1)) = \sum_{i=0}^{s-2} b^i = \frac{b^{s-1} - 1}{b - 1}$$

Dann erhalten wir

$$\begin{aligned}
 I(T(x, s)) &= 1 + a \cdot I(T(b^{s-1}, s-1)) + I(T(d, s-1)) \\
 &= 1 + a \cdot \frac{b^{s-1} - 1}{b-1} + \frac{d}{b-1} + s - 1 \\
 &= \frac{a \cdot b^{s-1} + d - a}{b-1} + s \\
 &\leq \frac{a \cdot b^{s-1} + d}{b-1} + s \\
 &= \frac{x}{b-1} + s
 \end{aligned}$$

Punkte: Pfadlänge [1]; Blätter [1,5]; innere Knoten [2,5]

Aufgabe 3 (4 + 4 Punkte)

Betrachten Sie einen neuen Rechner mit den Registern PC (Program Counter), ACC (Akkumulator), B (Operandenregister) und IN (Indexregister), wie in Abbildung 7 skizziert. Die Instruktionen des Rechners sind in Tabelle 3 zusammengestellt. Dabei werden die Register wieder durch 2 Bits kodiert, die in den Befehlen mit D und S bezeichnet werden. Die Operationscodes op und die Bedingungen c sind entsprechend des RETI Rechners gewählt.

- Zeigen Sie, dass die Instruktionen des neuen Rechners mit den Datenpfaden aus Abbildung 7 durchgeführt werden können.
- Können alle $LOAD$, $STORE$ und $COMPUTE$ -Befehle des RETI Rechners auf dem neuen Rechner simuliert werden? Geben Sie entsprechende Befehlsfolgen und die notwendigen Einschränkungen für den neuen Rechner an.

I[31:30]	I[29:28]	I[27]	I[26]	I[25:0]	Befehl	Wirkung
LOAD-Befehle: $D \in \{PC, IN, B, ACC\}$ $i \in \mathbb{B}^{26}$						
01	00		D	i	$LOAD D i$	$D := M(i)$ $PC := PC + 1$
01	01		D	*	$LOADIN D$	$D := M(IN)$ $PC := PC + 1$
01	10		D	i	$LOADI D i$	$D := i$ $PC := PC + 1$
STORE-Befehle: $S \in \{PC, IN\}$ $i \in \mathbb{B}^{26}$						
10	00	*	*	i	$STORE i$	$M(i) := ACC$ $PC := PC + 1$
10	10	*	*	*	$STOREIN$	$M(IN) := ACC$ $PC := PC + 1$
10	01		S	*	$MOVE S$	$ACC := S$ $PC := PC + 1$
COMPUTE-Befehle: $op \in \{SUB, ADD, OPLUS, OR, AND\}$						
00		op	*	*	op	$ACC := ACC op B$ $PC := PC + 1$

Tabelle 1: Instruktionen des neuen Rechners

Lösung:

- Siehe Abb. 8.
- Einschränkungen:
 - Nur ein Index-Register.
 - Das Operandenregister B kann nicht direkt auf den Adressbus *gelegt* werden.

- Man benötigt zur Simulation der RETI Befehle Hilfsspeicherzellen im Speicher.
- Simulation der nicht direkt implementierten RETI Befehle:

1) LOADIN1 D i:

MOVE IN	$; ACC = IN$
LOADI B i	$; B = i$
ADD	$; ACC = ACC + B = IN + i$
STORE 100	$; M(100) = ACC = IN + i$
LOAD IN 100	$; IN = M(100) = IN + i$
LOADIN D	$; D = M(IN) = M(IN + i)$

2) STOREIN1 D i:

STORE 100	$; M(100) = ACC$
MOVE IN	$; ACC = IN$
LOADI B i	$; B = i$
ADD	$; ACC = ACC + B = IN + i$
STORE 101	$; M(101) = ACC = IN + i$
LOAD IN 101	$; IN = M(101) = IN + i$
LOAD ACC 100	$; ACC = M(100) = ACC$
STOREIN	$; M(IN) = M(IN + i) = ACC$

3) MOVE S D:

MOVE S	$; ACC = S(PC, IN)$
STORE 100	$; M(100) = ACC$
LOAD D 100	$; D = M(100) = ACC = S$

4) Compute Immediate, Beispiel: ADDI D i:

LOADI B i	$; B = i$
ADD	$; ACC = ACC + B = ACC + i$
STORE 100	$; M(100) = ACC = ACC + B = ACC + i$
LOAD D 100	$; D = M(100) = ACC + i$

5) Compute Memory, Beispiel: ADD D i:

LOAD B i	$; B = M(i)$
ADD	$; ACC = ACC + B = ACC + M(i)$
STORE 100	$; M(100) = ACC = ACC + B = ACC + M(i)$
LOAD D 100	$; D = M(100) = ACC + M(i)$

Abgabe: 24. Januar 2014, 17⁰⁰ über das Übungsportal

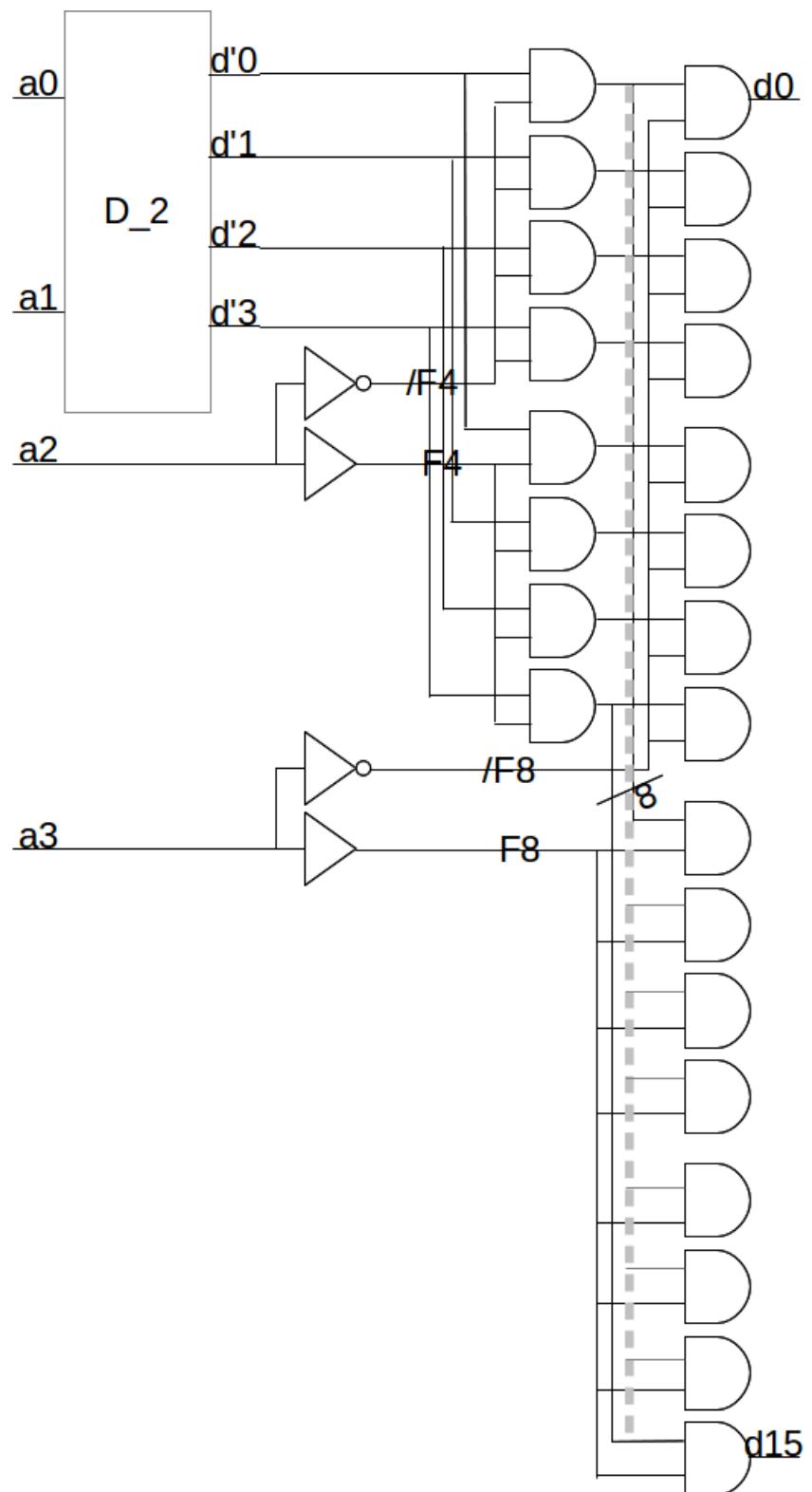


Abbildung 3: 4-Bit-Dekodierer

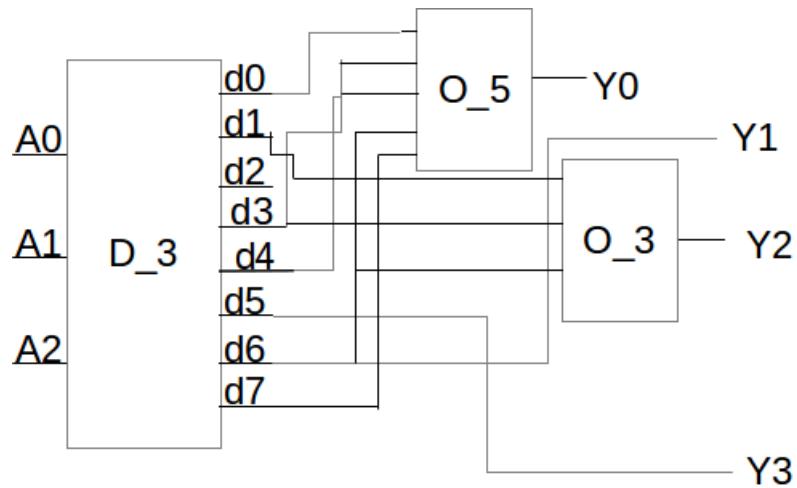


Abbildung 4: Schaltkreis für die Datum Funktion

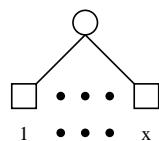


Abbildung 5: Induktionsvoraussetzung

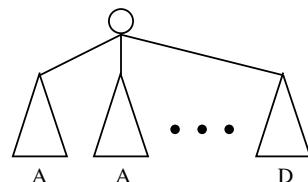


Abbildung 6: Induktionsschritt

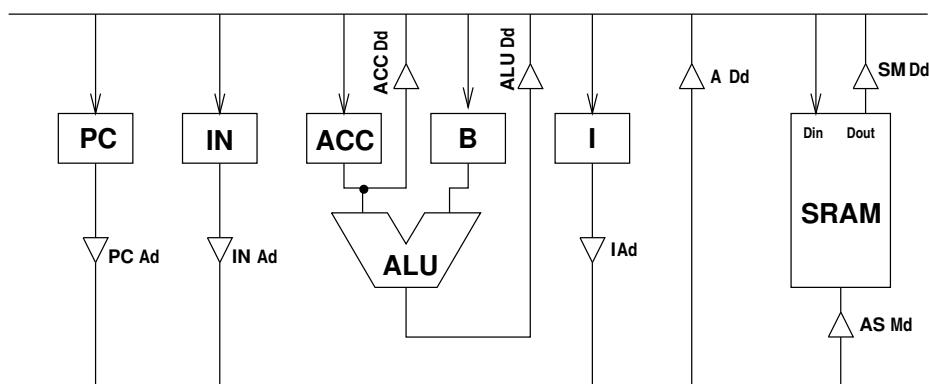


Abbildung 7: Datenpfade des neuen Rechners

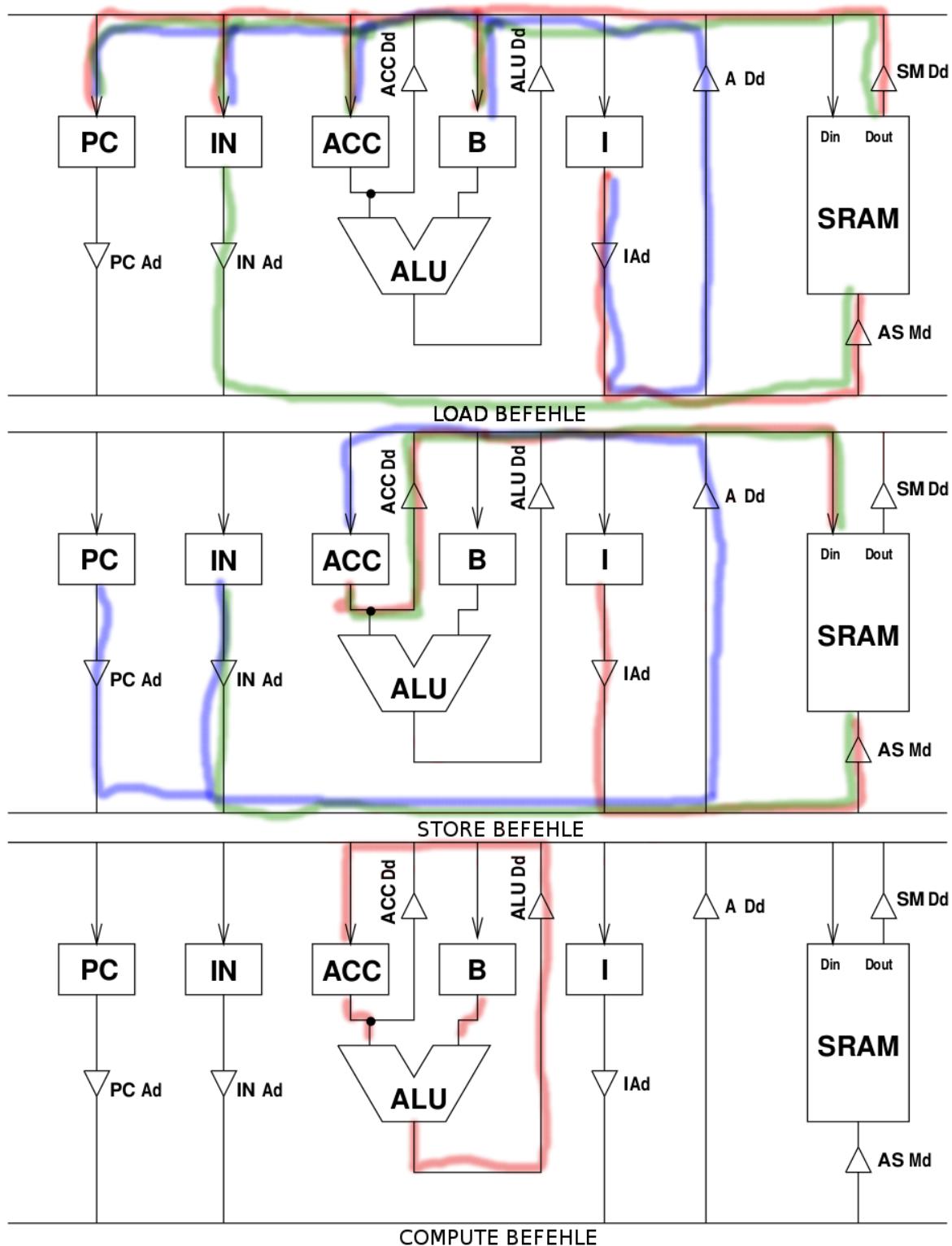


Abbildung 8: Datenpfade der Load, Store und Compute Befehle.

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 31. Januar 2014

Technische Informatik Übungsblatt 11

Aufgabe 1 (2 + 2 + (0 + 2 (Bonus)) + 2 + (0 + 2 (Bonus)) Punkte)

Für die ReTI benötigen wir eine Reihe von Kontrollsignalen, die die Register, Treiber, ALU und SRAM ansprechen (Siehe Kap. 4.5).

Erstellen Sie beispielhaft für die folgenden Kontrollsignale des ReTI Rechners die dazugehörigen *Boolesche Funktionen* für die Kontrollogik.

- Register
 - a) $ACCcken$ (Clock-Enable-Signal für den Akkumulator)
- Treiber (Enable-Signale)
 - b) $/ACCDdoe$ (Verbindung ACC mit Datenbus)
 - c) $/SMDdoe$ (Verbindung SRAM Ausgangstreiber mit Datenbus)
- ALU
 - d) $f[2:0]$ (Funktionauswahl der ALU. Codierung wie in Kap. 3.5)
 - e) $sext$ (Sign Extension)

Bestimmen Sie zunächst zu welchen Zeitpunkten und dann unter welchen Bedingungen (Befehlsart) die Signale aktiv sein müssen. Beachten Sie dabei, dass einige Signale active-low sind.

Hinweis: Es ist nicht nötig ein Polynom anzugeben, Sie können wie in der Vorlesung einen beliebigen Booleschen Ausdruck angeben. Geben Sie also Boolesche Ausdrücke für $ACCcken_{pre}$, $ACCDdoe_{pre}$, $SMDdoe_{pre}$ an. Wie in der Vorlesung erwähnt, sind $f[2 : 0]$ und $sext$ nicht Ausgänge eines FF der Kontrollogik, sondern rein kombinatorische Signale. Es genügt also, direkt entsprechende Boolesche Ausdrücke anzugeben, die zudem nicht von s_0 , s_1 , und E abhängen müssen.

Für die Signale $f[2:0]$ und $sext$ benötigen Sie keine Kodierung des Taktes.

Aufgabe 2 (1 + 3 + 2 Punkte)

- a) Zeichnen Sie ein RS-Flipflop, wie es in der Vorlesung vorgestellt wurde.
- b) Berechnen Sie die minimale Schreibpulsweite, damit ein Schreibvorgang am RS-Flipflop gelingt.
Hinweis: Die Zeit für das spikefreie Umschalten eines NAND-Gatters beträgt 0.41 ns
- c) Berechnen Sie die minimale und die maximale Verzögerungszeit des RS-Flipflops.

Auf den Webseiten der Veranstaltung finden Sie unter [Vorlesungsmaterial → Hilfsmaterial](#) eine kurze Übersicht über Verzögerungszeiten und Timing einiger wichtiger Gatter und Komponenten (“NanGate Bibliothek”), die sie u.a. für diese Aufgabe benötigen werden.

Aufgabe 3 (3 + 3 Punkte)

Betrachten Sie einen n -Bit-Conditional-Sum-Addierer, der aus EXOR-Gattern, 1-Bit-Multiplexern, AND-Gattern, OR-Gattern und Treibern der NanGate-Bibliothek aufgebaut ist. Auf den Webseiten der Veranstaltung finden Sie unter [Vorlesungsmaterial → Hilfsmaterial](#) eine kurze Übersicht über Verzögerungszeiten und Timing einiger wichtiger Gatter und Komponenten (“NanGate Bibliothek”), die sie u.a. für diese Aufgabe benötigen werden.

- a) Geben Sie die maximale Verzögerungszeit an, zu der das Ausgangscarry-Signal schalten kann, unter der Voraussetzung, dass die Inputs zur Zeit $t_0 = 0$ schalten können, aber nicht müssen. Sie können bei der Analyse des n -Bit-Conditional-Sum-Addierers zunächst Treiber vernachlässigen, die man eigentlich zum Vervielfältigen eines Signals auf mehr als 10 Eingänge benötigen würde.
- b) Geben Sie nun die maximale Verzögerungszeit an, zu der das Ausgangscarry-Signal eines 32-Bit-Conditional-Sum-Addierers schalten kann, jetzt aber unter Berücksichtigung der Tatsache, dass zum Vervielfältigen eines Signals auf mehr als 10 Eingänge Treiber benötigt werden.

Hinweis: Bei der Timing-Analyse des n -Bit-Conditional-Sum-Addierers stellen die Analyse von Volladdierern und n -Bit-Multiplexern natürlich notwendige Teilschritte dar. Ansonsten können Sie sich an der Analyse zur Bestimmung der Tiefe des CSA_n aus der Vorlesung orientieren. An der Stelle des Beitrags 1 eines Gatters zur Tiefe muss nun natürlich seine tatsächliche Verzögerungszeit stehen.

Aufgabe 4 (2 + 4 Punkte)

In der Vorlesung wurde eine Methode vorgestellt (Kap. 5.1, Folie 31ff, *Fall 2*), mit der sich bestimmen lässt, zu welchen Zeitpunkten an den Ausgängen einer Schaltung sicher ein stabiler Wert anliegt, wenn die Eingänge sich zu einem Zeitpunkt t_0 (bzgl. M) ändern, dabei aber unbekannt ist, welche Signale sich ändern und wie sie sich ändern. Vor und nach t_0 sind die Werte stabil.

Betrachten Sie die Schaltung in Abbildung 1.

Für den Inverter der NanGate-Bibliothek sind folgende Schaltzeiten (in ns) angegeben:

$$\tau_{PLH} = [0.01, 0.15]$$

$$\tau_{PHL} = [0.00, 0.08]$$

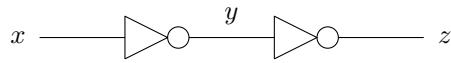


Abbildung 1: Zwei Inverter

- a) Berechnen Sie mit der in der Vorlesung angegebenen Methode die Intervalle, in denen die einzelnen Gatter schalten.

In welchem Zeitbereich kann – basierend auf der in der Vorlesung angegebenen Methode – der Ausgang der Schaltung spätestens schalten?

- b) Betrachten Sie nun getrennt voneinander sämtliche möglichen Schaltvorgänge am Eingang der Schaltung und berechnen Sie jeweils die Intervalle, in denen die Gatter schalten. Illustrieren Sie dies jeweils mit einem Timingdiagramm.

In welchem Zeitraum kann – basierend auf diesen Resultaten – der Ausgang der Schaltung schalten?

Begründen Sie den Unterschied zwischen den beiden Methoden.

Abgabe: 7. Februar 2014, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 31. Januar 2014

Technische Informatik Musterlösung zu Übungsblatt 11

Aufgabe 1 (2 + 2 + (0 + 2 (Bonus)) + 2 + (0 + 2 (Bonus)) Punkte)

Für die ReTI benötigen wir eine Reihe von Kontrollsignalen, die die Register, Treiber, ALU und SRAM ansprechen (Siehe Kap. 4.5, Folie 65).

Erstellen Sie beispielhaft für die folgenden Kontrollsgrade des ReTI Rechners die dazugehörigen *Boolesche Funktionen* für die Kontrollogik.

- Register
 - a) $ACCcken$ (Clock-Enable-Signal für den Akkumulator)
- Treiber (Enable-Signale)
 - b) $/ACCDdoe$ (Verbindung ACC mit Datenbus)
 - c) $/SMDdoe$ (Verbindung SRAM Ausgangstreiber mit Datenbus)
- ALU
 - d) $f[2:0]$ (Funktionauswahl der ALU. Codierung wie in Kap. 3.5)
 - e) $sext$ (Sign Extension)

Bestimmen Sie zunächst zu welchen Zeitpunkten und dann unter welchen Bedingungen (Befehlsart) die Signale aktiv sein müssen. Beachten Sie dabei, dass einige Signale active-low sind.

Hinweis: Es ist nicht nötig ein Polynom anzugeben, Sie können wie in der Vorlesung einen beliebigen Booleschen Ausdruck angeben. Geben Sie also Boolesche Ausdrücke für $ACCcken_{pre}$, $ACCDdoe_{pre}$, $SMDdoe_{pre}$ an. Wie in der Vorlesung erwähnt, sind $f[2 : 0]$ und $sext$ nicht Ausgänge eines FF der Kontrollogik, sondern rein kombinatorische Signale. Es genügt also, direkt entsprechende Boolesche Ausdrücke anzugeben, die zudem nicht von s_0 , s_1 , und E abhängen müssen.

Für die Signale $f[2:0]$ und $sext$ benötigen Sie keine Kodierung des Taktes.

Lösung:

- a) $ACCck$ Akkumulator wird geclocked bei

- allen LOAD-Befehlen mit D = ACC
- Move-Befehl mit D = ACC
- allen Compute-Befehlen mit D = ACC

Der Akkumulator soll geclockt werden im 3. Takt von Execute (siehe Abb. 1) (also bei $E = 1, s1 = 1, s0 = 0$).

Da wir einen Takt vorher das Signal generieren, erhalten wir: $E = 1, s1 = 0, s0 = 1$

Damit ergibt sich:

```
ACCck :=  (E * /s1 * s0          ; Startzeit bei P2 von E
           */I31 * /I30          ; typ = Compute
           *I25 * I24            ; D = ACC
           +E * /s1 * s0          ; Startzeit bei P2 von E
           */I31 * I30            ; typ = LOAD
           *I25 * I24            ; D=ACC
           +E * /s1 * s0          ; Startzeit bei P2 von E
           *I31 * /I30 * I29 * I28 ; typ = MOVE
           *I25 * I24)            ; D=ACC
```

b) $/ACCDoe$

- 1) $/ACCDoe$ muss aktiv sein bei STORE, STOREIN1 und STOREIN2.

Es soll aktiv werden im 1. Takt von Execute (siehe Abb. 1). Das Signal muss schon ein Takt vorher aktiv sein (also bei $E = 1, s1 = 0, s2 = 0$). Zudem muss es insgesamt 3 Takte aktiv sein.

Damit:

```
/ACCDoe :=  /(E * /s1 * /s0          ; Startzeit bei P0 von E
               *I31 * /I30 * /I29          ; typ = STORE und STOREIN1
               +E * /s1 * /s0            ; Startzeit bei P0 von E
               *I31 * /I30 * /I28          ; typ = STORE und STOREIN2
               +E * /s1 * /s0            ; P1 von E
               *I31 * /I30 * /I29          ; typ = STORE und STOREIN1
               +E * /s1 * /s0            ; P1 von E
               *I31 * /I30 * /I28          ; typ = STORE und STOREIN2
               +E * s1 * /s0            ; P2 von E
               *I31 * /I30 * /I29          ; typ = STORE und STOREIN1
               +E * s1 * /s0            ; P2 von E
               *I31 * /I30 * /I28)          ; typ = STORE und STOREIN2
```

c) $/SMDoe$

- 1) $/SMDoe$ muss aktiv sein bei Fetch, Compute Memory, LOAD, LOADINj.

Es soll aktiv werden von P1 bis P0 (siehe Abb. 1). Das Signal muss schon ein Takt vorher aktiv sein (also bei $s1 = 0, s2 = 0$). Zudem muss es insgesamt zwei Mal 3 Takte aktiv sein.

Damit:

```
/SMDoepre :=  (/E * /s1 + /E * s1 * /s0)+ ; Fetch
/SMDoepre :=  (E * /s1 + E * s1 * /s0)* ; Execute
          (/I31 * /I30 * I29+          ; typ = Compute Memory
           /I31 * I30 * /I29 * /I28+    ; typ = LOAD
           /I31 * I30 * /I29 * I28+    ; typ = LOADIN1
           /I31 * I30 * I29 * /I28)    ; typ = LOADIN2
```

d) $f[2 : 0]$:

- Bei Compute Befehlen ist $f[2 : 0] = I[28 : 26]$, bei allen anderen Befehlen soll die ALU addieren ($f[2 : 0] = 011$).
- $f[0] = /I31 * /I30 * I26 + (/I31 * /I30)$.
- $f[1] = /I31 * /I30 * I27 + (/I31 * /I30)$.
- $f[2] = /I31 * /I30 * I28$.

e) $sext$:

- Sign Extension wird bei all den Befehlen durchgeführt, bei denen der Parameter i als Zweierkomplementzahl interpretiert wird.
- Sign Extension kann bei LOAD, STORE und Compute-Memory Operationen durchgeführt werden, weil die obersten 16 Bit in diesen Fällen ignoriert werden.
- Sign Extension darf nicht bei LOADI und logischen Compute-Immediate Befehlen durchgeführt werden (in diesen Fällen wird mit 0^8 aufgefüllt).
- $\text{sext} = \begin{array}{ll} /I31 * I30 * (/I29 + /I28) & \text{LOAD und LOADINj} \\ +I31 * /I30 & \text{Alle STORE und MOVE} \\ +/I31 * /I30 * /I29 * /I28 & \text{Compute Immediate Arithmetisch} \\ +/I31 * /I30 * I29 & \text{Compute Memory} \\ +I31 * I30 & \text{JUMP} \end{array}$

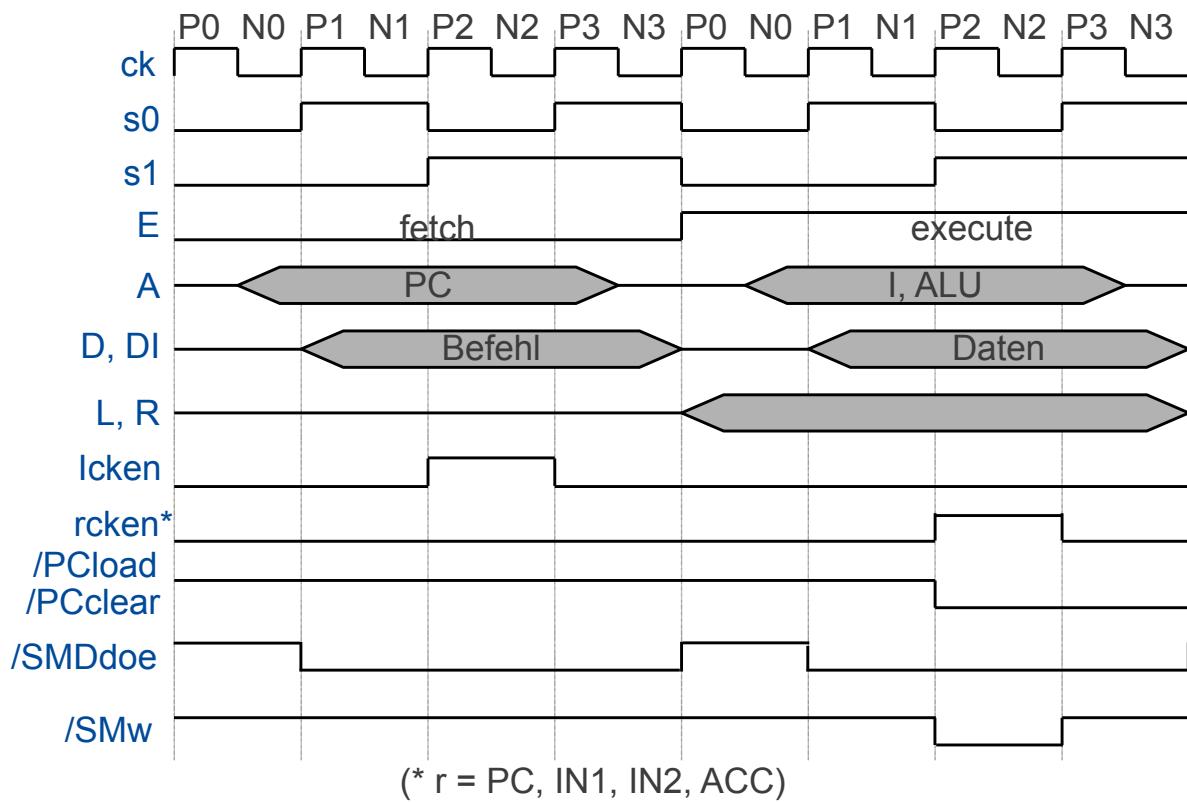


Abbildung 1: Timingdiagramm ReTI

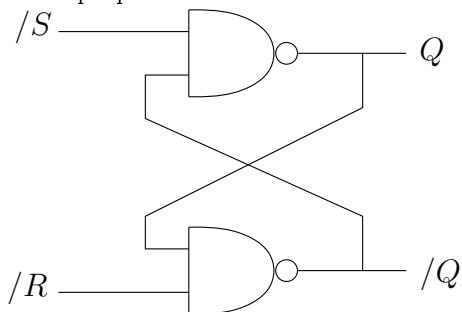
Aufgabe 2 (1 + 3 + 2 Punkte)

- Zeichnen Sie ein RS-Flipflop, wie es in der Vorlesung vorgestellt wurde.
- Berechnen Sie die minimale Schreibpulsweite, damit ein Schreibvorgang am RS-Flipflop gelingt.
Hinweis: Die Zeit für das spikefreie Umschalten eines NAND-Gatters beträgt 0.41 ns
- Berechnen Sie die minimale und die maximale Verzögerungszeit des RS-Flipflops.

Auf den Webseiten der Veranstaltung finden Sie unter **Vorlesungsmaterial** → **Hilfsmaterial** eine kurze Übersicht über Verzögerungszeiten und Timing einiger wichtiger Gatter und Komponenten (“NanGate Bibliothek”), die sie u.a. für diese Aufgabe benötigen werden.

Lösung:

- a) RS-Flipflop:



- b) Wegen der Symmetrie von $/S$ und $/R$ können wir uns auf $/S$ beschränken.

$$/S : 1 \rightarrow 0 \text{ zur Zeit } t_0.$$

$$Q : 0 \rightarrow 1 \text{ zur Zeit } t_1 := t_0 + t_{PLH}(\text{Nand}) = t_0 + (0.01, 0.15).$$

$$/Q : 1 \rightarrow 0 \text{ zur Zeit } t_2 := t_1 + t_{PHL}(\text{Nand}) = t_0 + (0.02, 0.27).$$

$$/S : 0 \rightarrow 1 \text{ zur Zeit } t_3 := t_0 + y \text{ (y Schreibpulsweite).}$$

Damit spikefreies Umschalten des oberen Nand-Gatters gewährleistet ist, muss $\min t_3 - \max t_2 \geq 0.41$ gelten, also:

$$\begin{aligned} \min t_3 - \max t_2 &\geq 0.41 \\ \Leftrightarrow t_0 + y - (t_0 + 0.27) &\geq 0.41 \\ \Leftrightarrow y &\geq 0.68 \end{aligned}$$

- c) Wie oben berechnet: $t_P(\text{RS - FF}) = (0.02, 0.27)$.

Aufgabe 3 (3 + 3 Punkte)

Betrachten Sie einen n -Bit-Conditional-Sum-Addierer, der aus EXOR-Gattern, 1-Bit-Multiplexern, AND-Gattern, OR-Gattern und Treibern der NanGate-Bibliothek aufgebaut ist. Auf den Webseiten der Veranstaltung finden Sie unter **Vorlesungsmaterial** → **Hilfsmaterial** eine kurze Übersicht über Verzögerungszeiten und Timing einiger wichtiger Gatter und Komponenten (“NanGate Bibliothek”), die sie u.a. für diese Aufgabe benötigen werden.

- a) Geben Sie die maximale Verzögerungszeit an, zu der das Ausgangscarry-Signal schalten kann, unter der Voraussetzung, dass die Inputs zur Zeit $t_0 = 0$ schalten können, aber nicht müssen. Sie können bei der Analyse des $n - Bit$ -Conditional-Sum-Addierers zunächst Treiber vernachlässigen, die man eigentlich zum Vervielfältigen eines Signals auf mehr als 10 Eingänge benötigen würde.
- b) Geben Sie nun die maximale Verzögerungszeit an, zu der das Ausgangscarry-Signal eines $32 - Bit$ -Conditional-Sum-Addierers schalten kann, jetzt aber unter Berücksichtigung der Tatsache, dass zum Vervielfältigen eines Signals auf mehr als 10 Eingänge Treiber benötigt werden.

Hinweis: Bei der Timing-Analyse des $n - Bit$ -Conditional-Sum-Addierers stellen die Analyse von Volladdierern und $n - Bit$ -Multiplexern natürlich notwendige Teilschritte dar. Ansonsten können Sie sich an der Analyse zur Bestimmung der Tiefe des CSA_n aus der Vorlesung orientieren. An der Stelle des Beitrags 1 eines Gatters zur Tiefe muss nun natürlich seine tatsächliche Verzögerungszeit stehen.

Lösung:

Man muss den längsten Pfad betrachten $(a, b, c_{-1}) \rightarrow (c_n, s)$.

- a) Verzögerung von a bzw. $b \rightarrow c_n$
 - b) Verzögerung von a bzw. $b \rightarrow s$
 - c) Verzögerung von $c_{-1} \rightarrow c_n$
 - d) Verzögerung von $c_{-1} \rightarrow s$
- a) $n - Bit$ -CSA
- Basisfall: $CSA_1 = FA$; Für die maximale Verzögerung nehmen wir den längsten Pfad vom MUX (Select zum Ausgang):
 - 1) $\text{exor} + \text{and} + \text{or} = 0.21 + 0.12 + 0.14 = \mathbf{0.47}$
 - 2) $2 \times \text{exor} = 2 \cdot 0.21 = 0.42$
 - 3) $\text{and} + \text{or} = 0.26$
 - 4) $\text{exor} = 0.21$ - $CSA_2 = CSA_1 + MUX = 0.47 + 0.16$
 - $CSA_{2^k} = CSA_1 + k \cdot MUX = 0.47 + 0.16$
- b) Bei einem $32 - Bit$ -Addierer müssen wir Treibern am MUX Ausgang des CSA_{16} hinzufügen.
- $$\begin{aligned} t_{out} &= t_0 + V(MUX) + V(TREIBER) + 4 \cdot V(MUX) + V(VA) \\ &= 0.16 + 0.11 + 4 \cdot 0.16 + 0.47 = 1.38 \end{aligned}$$

Hinweis: In der Vorlesung wurde ein Treiberbaum statt mehrere Treiber direkt am Ausgang eines Schaltkreis angeschlossen. Dies ist erlaubt, auch wenn in diesem Fall nicht nötig da der MUX Ausgang von einem Gatter gesteuert ist, daher kann er bereits ≤ 10 Eingänge steuern. Hier haben wir Treiber an den Inputs vernachlässigt.

Aufgabe 4 (2 + 4 Punkte)

In der Vorlesung wurde eine Methode vorgestellt (Kap. 5.1, Folie 31ff, *Fall 2*), mit der sich bestimmen lässt, zu welchen Zeitpunkten an den Ausgängen einer Schaltung sicher ein stabiler Wert anliegt, wenn die Eingänge sich zu einem Zeitpunkt t_0 (bzgl. M) ändern, dabei aber unbekannt ist, welche Signale sich ändern und wie sie sich ändern. Vor und nach t_0 sind die Werte stabil.

Betrachten Sie die Schaltung in Abbildung 2.

Für den Inverter der NanGate-Bibliothek sind folgende Schaltzeiten (in ns) angegeben:

$$\tau_{PLH} = [0.01, 0.15]$$

$$\tau_{PHL} = [0.00, 0.08]$$

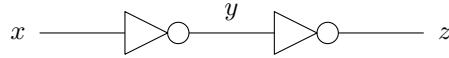


Abbildung 2: Zwei Inverter

- a) Berechnen Sie mit der in der Vorlesung angegebenen Methode die Intervalle, in denen die einzelnen Gatter schalten.

In welchem Zeitbereich kann – basierend auf der in der Vorlesung angegebenen Methode – der Ausgang der Schaltung spätestens schalten?

- b) Betrachten Sie nun getrennt voneinander sämtliche möglichen Schaltvorgänge am Eingang der Schaltung und berechnen Sie jeweils die Intervalle, in denen die Gatter schalten. Illustrieren Sie dies jeweils mit einem Timingdiagramm.

In welchem Zeitraum kann – basierend auf diesen Resultaten – der Ausgang der Schaltung schalten?

Begründen Sie den Unterschied zwischen den beiden Methoden.

Lösung:

- a) x schaltet zum Zeitpunkt t_0 bzgl. M

y schaltet zum Zeitpunkt $t_0 + (\min\{0.00, 0.01\}, \max\{0.08, 0.15\}) = t_0 + (0.00, 0.15)$ bzgl. M

z schaltet zum Zeitpunkt $t_0 + (0.00, 0.15) + (0.00, 0.15) = t_0 + (0.00, 0.30)$ bzgl. M

- b) 1) Fall 1: $x = 0 \rightarrow x = 1$

x schaltet zum Zeitpunkt t_0 bzgl. M

y schaltet zum Zeitpunkt $t_0 + t_{PHL}^{\text{NOT}} = t_0 + (0.00, 0.08)$ bzgl. M

z schaltet zum Zeitpunkt $t_0 + (0.00, 0.08) + t_{PLH}^{\text{NOT}} = t_0 + (0.00, 0.08) + (0.01, 0.15) = t_0 + (0.01, 0.23)$ bzgl. M

- 2) Fall 2: $x = 1 \rightarrow x = 0$

x schaltet zum Zeitpunkt t_0 bzgl. M

y schaltet zum Zeitpunkt $t_0 + t_{PLH}^{\text{NOT}} = t_0 + (0.01, 0.15)$ bzgl. M

z schaltet zum Zeitpunkt $t_0 + (0.01, 0.15) + t_{PHL}^{\text{NOT}} = t_0 + (0.01, 0.15) + (0.00, 0.08) = t_0 + (0.01, 0.23)$ bzgl. M

Zweite Methode ist exakter, da sie durch das Betrachten aller Fälle Zusammenhänge zwischen Signalwerten berücksichtigt. In dieser Schaltung kann es etwa nicht sein, daß beide Inverter von 0 auf 1 schalten und damit beide die maximale Verzögerung verursachen.

Abgabe: 7. Februar 2014, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 7. Februar 2014

Technische Informatik

Übungsblatt 12

Aufgabe 1 (6 Punkte)

Zeigen Sie, dass der Schreibvorgang bei dem in der Vorlesung vorgestellten D-Latch (siehe Abbildung 1) mit den Parameterwerten aus Tabelle 1 gelingt. Sie müssen dafür nachweisen, dass die angegebenen Setup-, Hold- und Verzögerungszeiten, sowie die Pulsweite korrekt sind.

Zur Erinnerung:

Ein *NAND*-Gatter schaltet spikefrei um, wenn der Abstand zwischen einer fallenden Flanke an einem Input des Gatters und einer steigende Flanke an dem zweiten Input des Gatters mindestens 0.41 ns ist.

Spikefreies Umschalten für ein RS-FlipFlop ist garantiert bei einer minimalen Pulsweite von 0.68 ns .

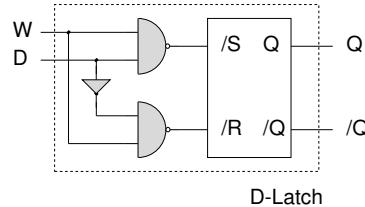


Abbildung 1: Aufbau eines D-Latches

Wichtig: Achten Sie auf eine nachvollziehbare Lösung, d. h. unter anderem, dass Sie die Bedeutung der auftretenden Zahlenwerte angeben sollten (also: für den Wert 0.12 ns sollte klar sein, dass es sich dabei um $t_{PLH}(\text{AND})$ handelt).

Symbol	Name	min	max
y	Pulsweite des Schreibimpulses	0.79	
t_{SDW}	Setup-Zeit von D bis W	0.49	
t_{HWD}	Hold-Zeit von D nach W	0.41	
t_{PWQ}	Verzögerungszeit von W bis Q	0.02	0.39

Tabelle 1: Parameterwerte des D-Latch in ns

Aufgabe 2 (10 Punkte)

Führen Sie in analoger Weise zur Timing-Analyse für Compute-Befehle aus der Vorlesung eine exakte Timing-Analyse der Fetch-Phase der ReTI durch.

Hinweise: Beginnen Sie dazu die Analyse bei P3 von execute als zeitlichem Bezugspunkt. Bei P3 von execute wird der Befehlszähler aktualisiert. Orientieren Sie sich bei Ihrer Analyse am idealisierten Timingdiagramm der ReTI. Dem idealisierten Timingdiagramm entnehmen Sie z.B., dass der Treiber PCAd bei N0 von fetch enabled wird. Leiten Sie nun eine Bedingung an die Zykluszeit her, die garantiert, dass die Setupzeit zum Abspeichern des Befehls im Instruktionsregister I bei P3 von fetch auf jeden Fall eingehalten wird.

Aufgabe 3 (6 + 2 Punkte)

- a) Schreiben Sie ein Programm für den RE-TI-Rechner, das die Fibonacci-Zahlen berechnet:

$$\begin{aligned}f(0) &= 0 \\f(1) &= 1 \\f(n) &= f(n - 1) + f(n - 2) \quad \text{für } n \geq 2\end{aligned}$$

Nehmen Sie dabei an, dass der Wert von n am Anfang in Speicherzelle M(99) steht. Am Ende Ihres Programms soll

in Speicherzelle M(100) $f(0)$ stehen,
in Speicherzelle M(101) $f(1)$ stehen,
...,
in Speicherzelle M(100+n) $f(n)$ stehen.

Geben Sie dazu zuerst ihren Algorithmus im Pseudocode an und kommentieren Sie ihr Assemblerprogramm. Für unkommentierte Programme werden keine Punkte vergeben!

Hinweis: Eine Liste der ReTI-Befehle finden Sie auf der Webseite der Vorlesung.

- b) Geben Sie nun den Zeitbedarf Ihres Programms in Abhängigkeit von n exakt an, unter der Voraussetzung, dass die Zykluszeit der ReTI 6.70 ns beträgt und die Abarbeitung eines Befehls 8 Zyklen benötigt. Wie ändert sich der Zeitbedarf Ihres Programms, wenn die Fetch-Phase (bei gleicher Zykluszeit) um einen Takt verkürzt werden kann?

Abgabe: 14. Februar 2014, 17⁰⁰ über das Übungsportal

Prof. Dr. Christoph Scholl
Dr. Paolo Marin

Freiburg, 7. Februar 2014

Technische Informatik

Musterlösung zu Übungsblatt 12

Aufgabe 1 (6 Punkte)

Zeigen Sie, dass der Schreibvorgang bei dem in der Vorlesung vorgestellten D-Latch (siehe Abbildung 1) mit den Parameterwerten aus Tabelle 1 gelingt. Sie müssen dafür nachweisen, dass die angegebenen Setup-, Hold- und Verzögerungszeiten, sowie die Pulsweite korrekt sind.

Zur Erinnerung:

Ein *NAND*-Gatter schaltet spikefrei um, wenn der Abstand zwischen einer fallenden Flanke an einem Input des Gatters und einer steigende Flanke an dem zweiten Input des Gatters mindestens 0.41 ns ist.

Spikefreies Umschalten für ein RS-FlipFlop ist garantiert bei einer minimalen Pulsweite von 0.68 ns .

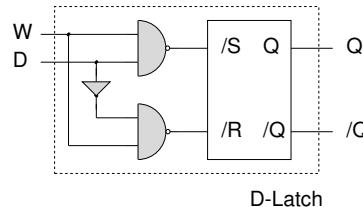


Abbildung 1: Aufbau eines D-Latches

Wichtig: Achten Sie auf eine nachvollziehbare Lösung, d. h. unter anderem, dass Sie die Bedeutung der auftretenden Zahlenwerte angeben sollten (also: für den Wert 0.12 ns sollte klar sein, dass es sich dabei um $t_{PLH}(\text{AND})$ handelt).

Symbol	Name	min	max
y	Pulsweite des Schreibimpulses	0.79	
t_{SDW}	Setup-Zeit von D bis W	0.49	
t_{HWD}	Hold-Zeit von D nach W	0.41	
t_{PWQ}	Verzögerungszeit von W bis Q	0.02	0.39

Tabelle 1: Parameterwerte des D-Latch in ns

Lösung:

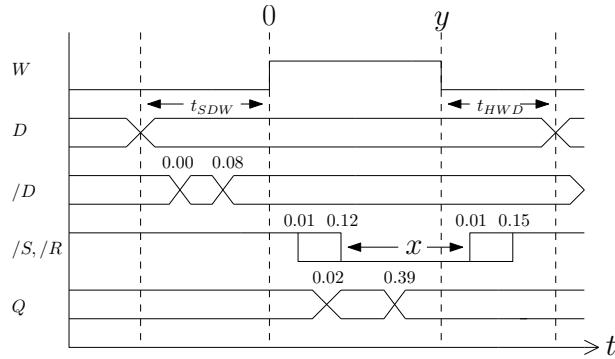


Abbildung 2: Timingdiagramm

Punkte: Je Parameterwert [2]; Wenn Lösung nicht nachvollziehbar, gibt es keine Punkte.

y :

- Sei $W = 1$ von $t = 0$ bis $t = y$.
- Entweder $/S$ oder $/R$ aktiv im Intervall $(0.01, 0.12)$ (wg. t_{PHL} des NAND) bis $y + (0.01, 0.15)$ (wg. t_{PLH} des NAND), d.h. der Schreibpuls am RS-FF ist min. von 0.12 ns bis $y + 0.01$ ns aktiv (in Abbildung 2 x).
- Pulsweite x des RS-FlipFlops muß laut Vorlesung ≥ 0.68 ns sein.
- Somit gilt: $y - 0.12 + 0.01 \geq 0.68 \Leftrightarrow y \geq 0.79$ ns.

t_{SDW} :

- Sei $W = 1$ zur Zeit $t = 0$ und vorher $W = 0$.
- Sei D stabil zur Zeit $-t_{SDW}$.
- Am oberen NAND kann nur ein Spike beim Übergang $(W, D) = (0, 1) \rightarrow (1, 0)$ entstehen. Um Spikefreiheit zu gewährleisten muss folglich gelten: $t_{SDW}^1 \geq 0.41$ ns.
- Am unteren NAND kann nur ein Spike beim Übergang $(W, \overline{D}) = (0, 1) \rightarrow (1, 0)$ entstehen. Dann liegt am Inverter eine fallende Flanke vor und somit: $t_{SDW}^2 \geq 0.41 + 0.08 = 0.49$ ns.
- $t_{SDW} \geq \max(t_{SDW}^1, t_{SDW}^2) = 0.49$ ns.

t_{HWD} :

- Nach Absenken von W muss D Spikefreiheit gewährleisten und somit $t_{HWD} \geq 0.41$ ns.

t_{PWQ} :

- Durch $W = 1$ wird entweder $/S$ oder $/R$ aktiv, an einem der beiden *vorderen* NANDs liegt somit eine fallende Flanke an.
- $t_{PWQ} = (0.01, 0.12) + (\min(t_{P/SQ}, t_{P/RQ}), \max(t_{P/SQ}, t_{P/RQ})) = (0.01, 0.12) + (0.01, 0.27) = (0.02, 0.39)$ ns.

Aufgabe 2 (10 Punkte)

Führen Sie in analoger Weise zur Timing-Analyse für Compute-Befehle aus der Vorlesung eine exakte Timing-Analyse der Fetch-Phase der ReTI durch.

Hinweise: Beginnen Sie dazu die Analyse bei P3 von execute als zeitlichem Bezugspunkt. Bei P3 von execute wird der Befehlszähler aktualisiert. Orientieren Sie sich bei Ihrer Analyse am idealisierten Timingdiagramm der ReTI. Dem idealisierten Timingdiagramm entnehmen Sie z.B., dass der

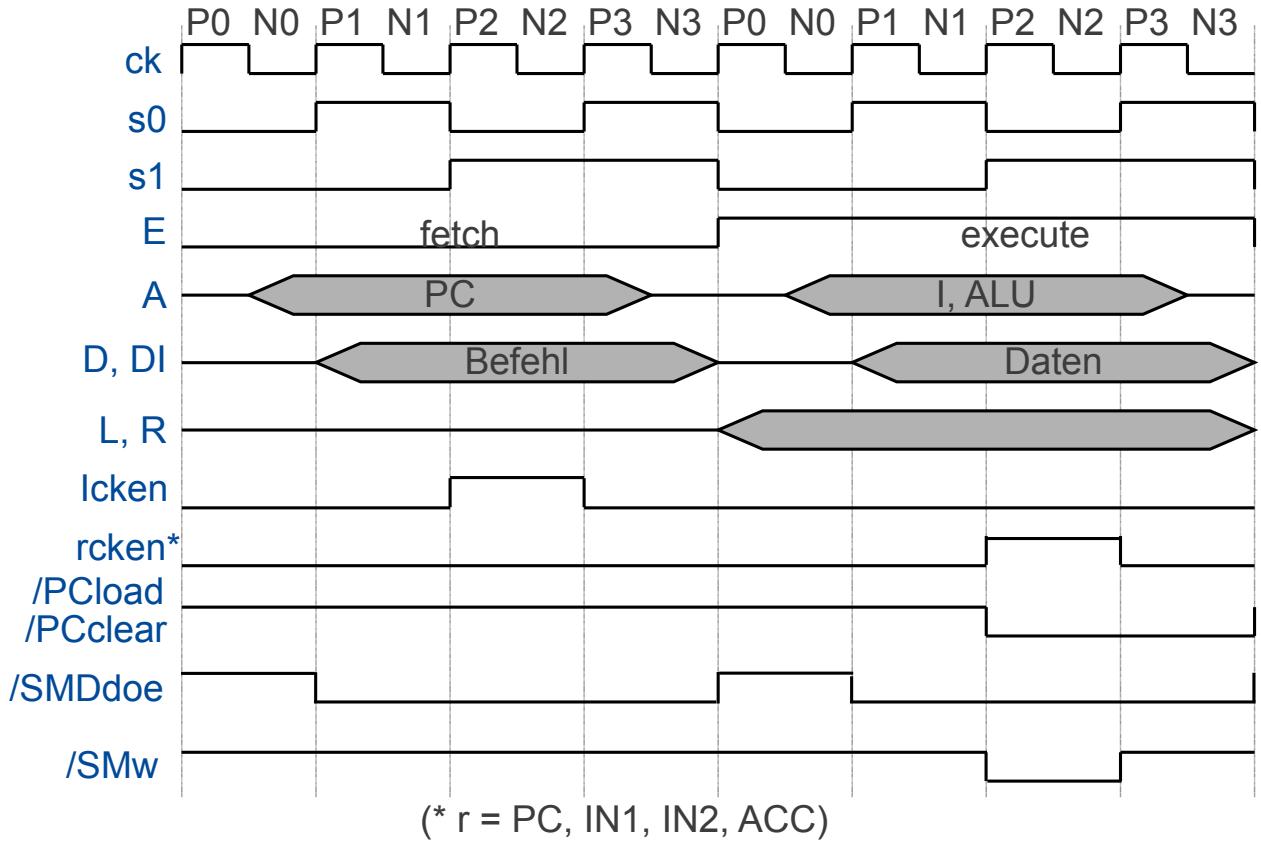


Abbildung 3: Timing von ReTI

Treiber PCAd bei N0 von fetch enabled wird. Leiten Sie nun eine Bedingung an die Zykluszeit her, die garantiert, dass die Setupzeit zum Abspeichern des Befehls im Instruktionsregister I bei P3 von fetch auf jeden Fall eingehalten wird.

Lösung:

- Beginn der Analyse bei $P3$ von Execute als zeitlicher Bezugspunkt.
- Bei $P3$ von Execute wird der Befehlszähler aktualisiert.
- Bei $P3$ wird der PC Zähler aktualisiert. In schlimmsten Fall kommt in Zähler (ein Register von steigender Flanke gesteuert) ein neuer Wert.
PC Ausgang gültig bei $\tau_1 = [0.12, 0.26]$
- Bei $N0$ von Fetch sind PCAd, /PCAdoe aktiv zur Zeit
 $\tau_2 = t_c + \tau_{p,al}^- = t_c + t_c/2 + [0.13, 0.56]$
- PC schon gültig vor Aktivierung von PCAd, falls $\max(\tau_1) \leq \min(\tau_2) \Leftrightarrow 0.26 \leq 3/2t_c + 0.13 \Leftrightarrow t_c \geq 0.09$
- A gültig zur Zeit $\tau_3 = \tau_2 + [0.03 + 0.11] = 3/2t_c + [0.16, 0.67]$ (Treiber)
- ASMD ist immer enabled, nur Treiber-Verzögerung berücksichtigen
 $\tau_4 = \tau_3 + [0.02 + 0.10] = 3/2t_c + [0.18, 0.77]$
- Lesezugriff am Speicherausgang $\tau_5 = \tau_4 + [0.0, 12.0] = 3/2t_c + [0.18, 12.77]$
Nur korrekt, wenn /SMDdoe rechtzeitig enabled!

- /SMDdoe aktiviert zur Zeit $\tau' = 2 \cdot t_c + \tau_{p,al}^+ = 2 \cdot t_c + [0.12, 0.41]$
- Daten am Speicherausgang aufgrund Treiber-Enable gültig zur Zeit $\tau'' = \tau' + [0.0, 7.0] = 2 \cdot t_c + [0.12, 7.41]$
- Daten am Speicherausgang gültig spätestens zur Zeit $\tau''' = \max(\max(\tau_5), \max(\tau''))$
- Bedingung für $\max(\max(\tau_5), \max(\tau''))$:
 $3/2t_c + 12.77 \geq 2 \cdot t_c + 7.41$
 $1/2t_c \leq 5.36 \Leftrightarrow t_c \leq 10.72$ Diesen Wert nehmen wir als minimale Taktperiode an und rechnen mit $\max(\tau_5)$ weiter.
- Daten müssen um die Setup-Zeit früher als P3 von Fetch am Eingang vom Register I:
 $\tau_6 = 4 \cdot t_c - t_{SDC} \geq \max(\tau_5) \Leftrightarrow 4 \cdot t_c - 0.08 \geq 5/2t_c + 12.77 \Leftrightarrow 5/2t_c \geq 12.85 \Leftrightarrow t_c \geq 5.14$
Es wäre nicht möglich, die Fetch-Phase um 1 Takt zu verkürzen.

Aufgabe 3 (6 + 2 Punkte)

- a) Schreiben Sie ein Programm für den RE-TI-Rechner, das die Fibonacci-Zahlen berechnet:

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(n) &= f(n-1) + f(n-2) \quad \text{für } n \geq 2 \end{aligned}$$

Nehmen Sie dabei an, dass der Wert von n am Anfang in Speicherzelle M(99) steht. Am Ende Ihres Programms soll
in Speicherzelle M(100) f(0) stehen,
in Speicherzelle M(101) f(1) stehen,
...,
in Speicherzelle M(100+n) f(n) stehen.

Geben Sie dazu zuerst ihren Algorithmus im Pseudocode an und kommentieren Sie ihr Assemblerprogramm. Für unkommentierte Programme werden keine Punkte vergeben!

Hinweis: Eine Liste der ReTI-Befehle finden Sie auf der Webseite der Vorlesung.

- b) Geben Sie nun den Zeitbedarf Ihres Programms in Abhängigkeit von n exakt an, unter der Voraussetzung, dass die Zykluszeit der ReTI 6.70 ns beträgt und die Abarbeitung eines Befehls 8 Zyklen benötigt. Wie ändert sich der Zeitbedarf Ihres Programms, wenn die Fetch-Phase (bei gleicher Zykluszeit) um einen Takt verkürzt werden kann?

Lösung:

- a)
- ```

0: LOADI ACC 0
1: STORE 100
2: LOADI ACC 1
3: STORE 101

4: LOAD ACC 99
5: ADDI ACC 99
6: STORE 99

```

```
7: LOADI ACC 100
8: MOVE ACC IN2
```

```
9: MOVE IN2 ACC
10: SUB ACC 99
11: JUMP= 11
```

```
13: MOVE IN2 ACC
14: STORE 98
```

```
15: ADDI IN2 1
16: LOADIN2 IN1 0
17: ADD IN1 98
18: SUBI IN1 100
19: MOVE IN1 ACC
20: STOREIN2 1
```

```
21: JUMP -12
22: ENDE
```

**Erläuterungen:**

In Speicherzelle  $M(99)$  wird  $n$  abgelegt. In den Zeilen 0 bis 3 wird  $M(100)$  mit und  $f_0 = 0$  und  $M(101)$  mit  $f_1 = 1$  initialisiert.

In den Zeilen 4 bis 6 wird  $n$  mit 99 addiert.

In den Zeilen 7 und 8 wird  $IN2$  mit 100 (Adresse von  $f_0$ ) initialisiert.

Die Zeilen 9 bis 11 prüfen den Wert von  $n$ . Wenn er gleich 0 ist ( $n' = n + 100$ , wird mit dem Inhalt von  $IN2$  geglichen), wird das Programm beendet.

In den Zeilen 13 bis 14 wird der Wert vom Zähler in Speicherzelle  $M(98)$  abgelegt.

In der Zeile 15 wird der Zähler inkrementiert.

In den Zeilen 16 bis 20 wird in  $IN1$  der Wert in  $M(IN2)$  kopiert, und mit dem Wert in  $M(98)$  addiert. Dann um 100 dekrementiert. Das Resultat in ACC und danach in  $M(IN2 + 1)$  kopiert.

- b)  $(12 + (n - 1) \cdot 12) \cdot 8 \cdot 6.70ns$

Bei einer Fetch-Phase von 3 statt 4 Taktten wird die Laufzeit um 7/8 verkürzt.

**Abgabe: 14. Februar 2014, 17<sup>00</sup> über das Übungsportal**

Prof. Dr. Christoph Scholl  
Dr. Paolo Marin

Freiburg, 31. Januar 2014

## Technische Informatik

### Datenblatt NanGate Bibliothek

Alle Zeiten in  $ns$ . Die Zeiten basieren auf den Annahmen:  $\delta \leq 0.13\ ns$  und Fanout  $\leq 10$

|              | AND  |      | OR   |      | XOR  |      | Treiber |      |
|--------------|------|------|------|------|------|------|---------|------|
|              | min  | max  | min  | max  | min  | max  | min     | max  |
| $\tau_{PLH}$ | 0.02 | 0.12 | 0.02 | 0.11 | 0.02 | 0.21 | 0.02    | 0.10 |
| $\tau_{PHL}$ | 0.02 | 0.12 | 0.04 | 0.14 | 0.01 | 0.14 | 0.02    | 0.11 |

|              | NAND |      | NOR  |      | XNOR |      | NOT  |      |
|--------------|------|------|------|------|------|------|------|------|
|              | min  | max  | min  | max  | min  | max  | min  | max  |
| $\tau_{PLH}$ | 0.01 | 0.15 | 0.01 | 0.14 | 0.02 | 0.14 | 0.01 | 0.15 |
| $\tau_{PHL}$ | 0.01 | 0.12 | 0.00 | 0.05 | 0.01 | 0.11 | 0.00 | 0.08 |

Tabelle 1: Verzögerungszeiten Grundgatter von NanGate

|              | Tristate                          |  | Treiber |      | Inv-Treiber |      |
|--------------|-----------------------------------|--|---------|------|-------------|------|
|              | Bezeichnung                       |  | min     | max  | min         | max  |
| $\tau_{PZL}$ | Enable-Zeiten                     |  | 0.03    | 0.10 | 0.01        | 0.08 |
| $\tau_{PZH}$ | Enable-Zeiten                     |  | 0.03    | 0.11 | 0.01        | 0.13 |
| $\tau_{PLZ}$ | Disable-Zeiten                    |  | 0.03    | 0.11 | 0.01        | 0.13 |
| $\tau_{PHZ}$ | Disable-Zeiten                    |  | 0.03    | 0.10 | 0.01        | 0.08 |
| $\tau_{PLH}$ | Umschaltverzögerung bei $/OE = 0$ |  | 0.02    | 0.07 | 0.02        | 0.14 |
| $\tau_{PHL}$ | Umschaltverzögerung bei $/OE = 0$ |  | 0.03    | 0.10 | 0.01        | 0.07 |

Tabelle 2: Verzögerungszeiten Tristate von NanGate

|                | <b>Multiplexer</b><br>Bezeichnung                | min  | max  |
|----------------|--------------------------------------------------|------|------|
| $\tau_{PDQLH}$ | Verzögerung Daten-Eingänge $D$ bis Ausgang $Q$   | 0.03 | 0.13 |
| $\tau_{PDQHL}$ | Verzögerung Daten-Eingänge $D$ bis Ausgang $Q$   | 0.05 | 0.15 |
| $\tau_{PSQLH}$ | Verzögerung Select-Eingang $sel$ bis Ausgang $Q$ | 0.02 | 0.16 |
| $\tau_{PSQHL}$ | Verzögerung Select-Eingang $sel$ bis Ausgang $Q$ | 0.04 | 0.14 |

Tabelle 3: Verzögerungszeiten Multiplexer (1-Bit-Multiplexer) von NanGate

|              | <b>D-Flipflop</b><br>Bezeichnung | min  | max  |
|--------------|----------------------------------|------|------|
| $\tau_{SDC}$ | Setupzeit von D bis CLK          | 0.08 |      |
| $\tau_{HCD}$ | Holdzeit von D nach CLK          | 0.14 |      |
| $\tau_{PCQ}$ | Verzögerungszeit von CLK bis Q   | 0.12 | 0.26 |
| $\tau_{PDQ}$ | Verzögerungszeit von D bis Q     | 0.10 | 0.21 |

Tabelle 4: Zeiten für D-FF von NanGate

|                | <b>RS-Flipflop</b><br>Bezeichnung | min  | max  |
|----------------|-----------------------------------|------|------|
| $x$            | Pulsweite                         | 0.68 |      |
| $\tau_{P/SQ}$  | Verzögerungszeit von /S bis Q     | 0.01 | 0.15 |
| $\tau_{P/S/Q}$ | Verzögerungszeit von /S nach /Q   | 0.02 | 0.27 |
| $\tau_{P/RQ}$  | Verzögerungszeit von /R bis Q     | 0.02 | 0.27 |
| $\tau_{P/R/Q}$ | Verzögerungszeit von /R bis /Q    | 0.01 | 0.15 |

Tabelle 5: Zeiten für RS-FF

|              | <b>D-Latch</b><br>Bezeichnung | min  | max  |
|--------------|-------------------------------|------|------|
| $y$          | Pulsweite des Schreibpulses   | 0.79 |      |
| $\tau_{SDW}$ | Setupzeit von D bis W         | 0.49 |      |
| $\tau_{HWD}$ | Holdzeit von D nach W         | 0.41 |      |
| $\tau_{PWQ}$ | Verzögerungszeit von W bis Q  | 0.02 | 0.39 |
| $\tau_{PDQ}$ | Verzögerungszeit von D bis Q  | 0.02 | 0.54 |

Tabelle 6: Zeiten für D-Latch

|            | <b>CY7C1079DV33</b><br>Bezeichnung | min  | max |
|------------|------------------------------------|------|-----|
| $t_{acc}$  | Lesezugriffszeit                   | 12.0 |     |
| $t_{OED}$  | Zeit von $/SMDdoe = 0$ bis $D$     | 7.0  |     |
| $t_{OEZ}$  | Zeit von $/SMDdoe = 1$ bis high-Z  |      | 7.0 |
| $t_{wc}$   | Schreibzykluszeit                  | 12.0 |     |
| $t_{SAW}$  | Setupzeit von $A$ bis $W$          | 0.0  |     |
| $t_{SAW}$  | Setupzeit von $A$ bis Ende $W$     | 9.0  |     |
| $t_{HWA}$  | Holdzeit von $A$ nach $W$          | 0.0  |     |
| $w$        | Schreibpulsweite                   | 9.0  |     |
| $t_{SDEW}$ | Setupzeit von $D$ bis Ende $W$     | 7.0  |     |
| $t_{HWD}$  | Holdzeit von $D$ nach $W$          | 0.0  |     |

Tabelle 7: Zeiten für Cypress SRAM

Prof. Dr. Christoph Scholl  
Dr. Paolo Marin

Freiburg, 30. Januar 2014

# Testat

## Technische Informatik

**Name:** \_\_\_\_\_

**Matrikel-Nr.:** \_\_\_\_\_

**Umfang:** 24 Seiten

**Bearbeitungszeit:** 90 Minuten

**Erlaubte Hilfsmittel:** Keine

**Übungsgruppe:** \_\_\_\_\_

Bitte prüfen Sie, ob Sie **alle Aufgabenblätter** erhalten haben und tragen Sie auf **allen** verwen-deten Blättern (auch den zusätzlich ausgeteilten) Ihren **Namen** und Ihre **Matrikelnummer** ein. Blätter ohne diese Information werden nicht berücksichtigt.

| Aufgabe | Punktzahl möglich | Punktzahl erreicht |
|---------|-------------------|--------------------|
| 1       | 6                 |                    |
| 2       | 12                |                    |
| 3       | 8                 |                    |
| 4       | 9                 |                    |
| 5       | 12                |                    |
| 6       | 19                |                    |
| 7       | 8                 |                    |
| 8       | 8                 |                    |
| 9       | 8                 |                    |
| Summe   | 90                |                    |

Das Erreichen von **40** Punkten wren hinreichend zum Bestehen, wrde es sich bei dem Übungstestat um eine echte Klausur handeln.

Dieses Übungstestat dient Ihnen zur Vorbereitung auf die Abschlussklausur. Die Aufgaben sind vergleichbar mit einer realen Klausur, sowohl in Hinblick auf die Schwierigkeit als auch auf die Länge.

Das Testat zählt nicht zum Zulassungskriterium. Die angegebenen Punkte dienen lediglich zur Orientierung. Würde es sich hier um eine echte Klausur handeln, wären **40 Punkte** hinreichend zum Bestehen.

Sie dürfen das Übungstestat mit einem **Zeitlimit von 90 Minuten** und **ohne Hilfsmittel**, um Klausurbedingungen zu simulieren.

Nächste Woche werden in den Übungen die Lösungen der Aufgaben vorgestellt.

## Aufgabe 1 (6 Punkte)

Sei  $\mathcal{B} = (\mathbf{M}, \cdot, +, \bar{\cdot})$  eine Boolesche Algebra. Beweisen Sie die folgende Beziehung:

$$\forall x_1, x_2 \in M : \overline{x_1 \cdot x_2 + x_1} \cdot \overline{x_2} = x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2}$$

Benutzen Sie für den Beweis nur die im folgenden angegebenen Axiome und Regeln der Booleschen Algebra und geben Sie in jedem Schritt an, welches Axiom Sie benutzen. Ausnahme: Kommutativität und Assoziativität müssen nicht angegeben werden. Ein “Beweis” mit einer Funktionstabelle ist nicht zulässig, da diese Beziehung für beliebige Boolesche Algebren gilt.

- |                            |                                              |                                               |
|----------------------------|----------------------------------------------|-----------------------------------------------|
| (i) Kommutativitat        | $a + b = b + a$                              | $a \cdot b = b \cdot a$                       |
| (ii) Assoziativitat       | $a + (b + c) = (a + b) + c$                  | $a \cdot (b \cdot c) = (a \cdot b) \cdot c$   |
| (iii) Absorption           | $a + (a \cdot b) = a$                        | $a \cdot (a + b) = a$                         |
| (iv) Distributivitat      | $a + (b \cdot c) = (a + b) \cdot (a + c)$    | $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ |
| (v) Komplementregel        | $a + (b \cdot \bar{b}) = a$                  | $a \cdot (b + \bar{b}) = a$                   |
| (vi) de-Morgan             | $\overline{(a + b)} = \bar{a} \cdot \bar{b}$ | $\overline{(a \cdot b)} = \bar{a} + \bar{b}$  |
| (vii) Doppeltes Komplement | $\bar{\bar{a}} = a$                          |                                               |

## Ihre Lösung zu Aufgabe 1:

**Aufgabe 2** (6 + 2 + 2 + 2 Punkte)

Gegeben sei ein Alphabet  $A$  mit  $A = \{a, b, c, d, e, f\}$ . Die Zeichen des Alphabets  $A$  treten mit folgenden Wahrscheinlichkeiten in Nachrichten auf:

$$P(a) = 0.46, P(b) = 0.30, P(c) = 0.08, P(d) = 0.07, P(e) = 0.05, P(f) = 0.04$$

Für die Datenübertragung von Nachrichten über dem Alphabet  $A$  soll eine Huffman-Kodierung verwendet werden.

- a) Konstruieren Sie einen entsprechenden Huffman-Baum. Ordnen Sie hierfür die Blätter von links nach rechts mit absteigender Häufigkeit an. Markieren Sie die linken Kanten mit 0 und die rechten Kanten mit 1.
  - b) Geben Sie das Huffman-Codewort  $c(x)$  für jedes Zeichen  $x \in A$  an.
  - c) Kodieren Sie die Nachricht  $baff$  mit Ihrer Huffman-Kodierung. Zur besseren Lesbarkeit fügen Sie bitte jeweils ein Komma zwischen die einzelnen Codewörter ein.
  - d) Betrachten Sie folgenden Code  $c$  über das Alphabet  $B = \{u, v, w, x, y, z\}$ :

|           |        |
|-----------|--------|
| $a \in B$ | $c(a)$ |
| u         | 111    |
| v         | 01     |
| w         | 1101   |
| x         | 1100   |
| y         | 0      |
| z         | 10     |

Kann es sich hierbei um einen Huffman-Code handeln? Begründen Sie Ihre Antwort.

## Ihre Lösung zu Aufgabe 2:

Name: \_\_\_\_\_

Matrikel-Nr.: \_\_\_\_\_

5

Ihre Lösung zu Aufgabe 2 (Fortsetzung):

A large rectangular grid consisting of 20 columns and 25 rows of small squares, intended for handwriting practice or continuation of a solution.

**Aufgabe 3** (2 + 8 Punkte)

Sei  $d = d_n d_{n-1} \dots d_0$  eine Festkommazahl mit  $n+1$  Vorkomma- und 0 Nachkommastellen in *Einer-Komplement-Darstellung*.

- a) Geben Sie die Interpretationsfunktion  $[\cdot]_1: \mathbb{B}^{n+1} \rightarrow \mathbb{N}$  für Einer-Komplement-Zahlen mit  $n+1$  Vorkomma- und 0 Nachkommastellen an.
- b) Zeigen Sie formal: Einer-Komplement-Zahlen mit  $n+1$  Vorkomma- und 0 Nachkommastellen können negiert werden, indem man jedes Bit komplementiert.

Ihre Lösung zu Aufgabe 3:

**Aufgabe 4** (3 + 6 Punkte)

Eine *topologische Sortierung* der Knoten eines gerichteten Graphen  $G = (V, E)$  wird definiert als eine bijektive Abbildung

$$tsort : V \mapsto \{1, \dots, |V|\},$$

für die gilt:

$$\forall v \in V : (\forall w \in V : ((w, v) \in E \Rightarrow tsort(w) < tsort(v)))$$

Eine topologische Sortierung eines Schaltkreises ist eine topologische Sortierung der Knoten des Schaltkreis-Graphens.

- Zu einem azyklischen, gerichteten Graphen  $G = (V, E)$  existiere die topologische Sortierung  $tsort_1$ . Ist diese eindeutig? (Beweis oder Gegenbeispiel)
- Zeigen Sie: Für jeden azyklischen, gerichteten Graphen  $G = (V, E)$  existiert eine topologische Sortierung.

*Hinweis:* Zeigen Sie zuerst, dass gilt:

$$G = (V, E) \text{ azyklisch} \Rightarrow \exists v \in V : \text{indeg}(v) = 0.$$

Beweisen sie anschliessend die Korrektheit der Aussage durch Induktion über die Anzahl von Knoten des Graphen.

**Ihre Lösung zu Aufgabe 4:**

Ihre Lösung zu Aufgabe 4 (Fortsetzung):

A large rectangular grid consisting of 20 columns and 25 rows of small squares, intended for handwritten solutions.

**Aufgabe 5** (5 + 2 + 3 + 2 Punkte)

Gegeben sei der Schaltkreis  $SK_1$  wie folgt:

$SK_1 := (X_3, G, typ, IN, Y_2)$ , wobei

$$X_3 = (x_1, x_2, x_3)$$

$$Y_2 = (v_4, v_2)$$

$$G = (V, E)$$

$$V = \{x_1, x_2, x_3\} \cup \{v_1, v_2, v_3, v_4, v_5\}$$

$$E = \{(v_1, v_2), (v_3, v_2), (x_1, v_1), (x_2, v_1), (x_1, v_5), (x_2, v_5), (v_5, v_3), (x_3, v_3), (v_5, v_4), (x_3, v_4)\}$$

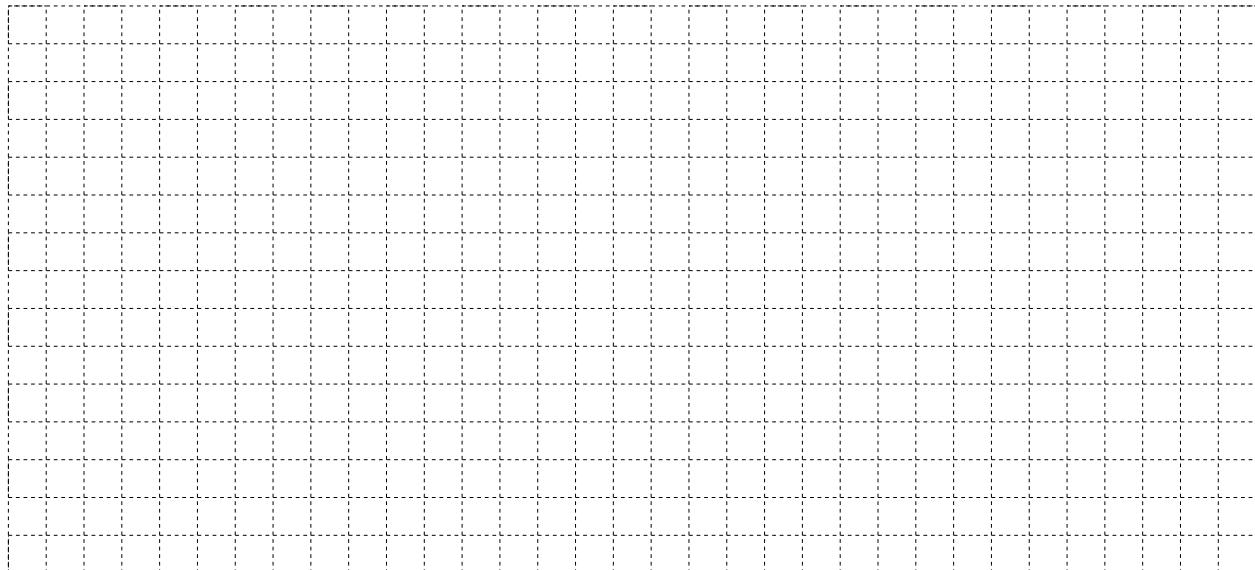
$$typ = \{(v_1 \mapsto and_2), (v_2 \mapsto or_2), (v_3 \mapsto and_2), (v_4 \mapsto exor_2), (v_5 \mapsto exor_2)\}$$

$$IN = \{(v_1 \mapsto ((x_1, v_1), (x_2, v_1))), (v_2 \mapsto ((v_1, v_2), (v_3, v_2))), \\ (v_3 \mapsto ((v_5, v_3), (x_3, v_3))), (v_4 \mapsto ((v_5, v_4), (x_3, v_4))), ($$

Hierbei wurde, anders als in der Vorlesung, eine verkürzte Schreibweise für die Kanten gewählt: wir definieren die Kantenmenge als  $E \subseteq V \times V$ . Gilt für eine Kante  $e = (v_i, v_j)$ , so ist  $Q(e) = v_i$  und  $Z(e) = v_j$ .

- a) Zeichnen Sie  $SK_1$ .
  - b) Geben Sie für  $SK_1$  eine topologische Sortierung an.
  - c) Führen Sie für  $SK_1$  eine symbolische Simulation durch.
  - d) Bestimmen Sie die Kosten sowie die Tiefe von  $SK_1$ .

## Ihre Lösung zu Aufgabe 5:



Ihre Lösung zu Aufgabe 5 (Fortsetzung):

A large rectangular grid consisting of 20 columns and 25 rows of small squares, intended for handwritten solutions.

**Aufgabe 6** (6 + 3 + 8 + 2 Punkte)

Betrachten Sie den PLA in Abbildung 1, der die beiden Funktionen  $f_1, f_2 \in \mathbb{B}_4$  realisiert. Inverter sind in dieser Abbildung als schwarze Punkte dargestellt.

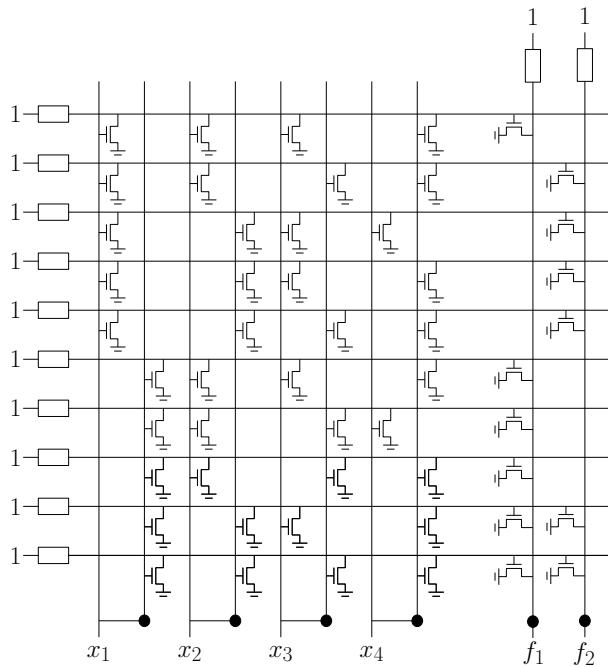
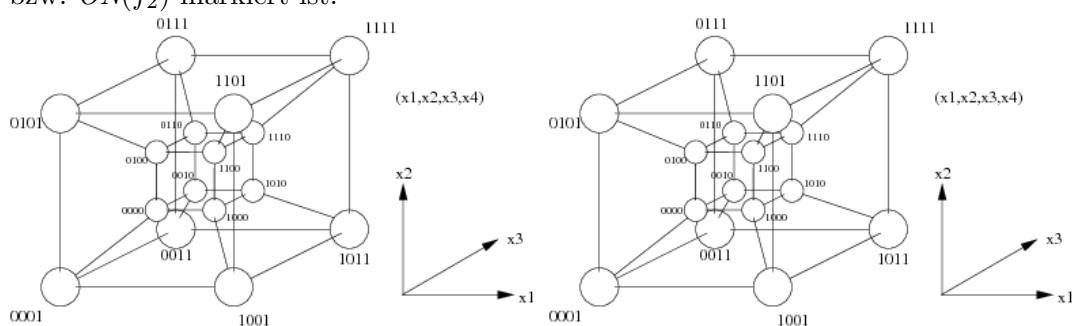


Abbildung 1: Ein PLA

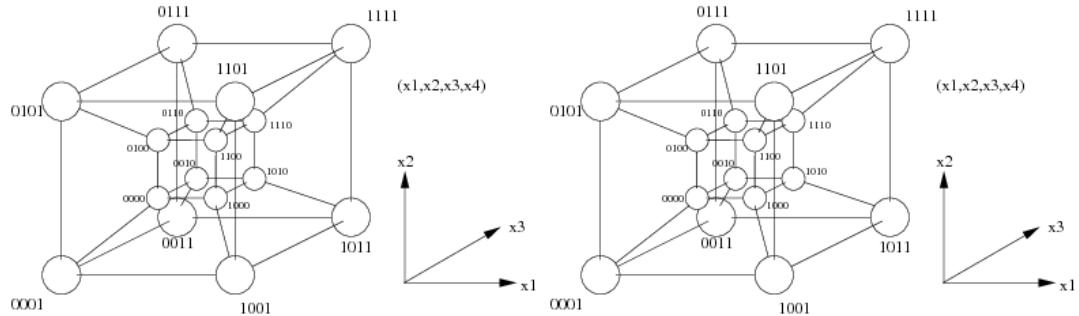
- a) Erstellen Sie die Polynome  $p_1$  und  $p_2$ , die durch die PLA-Realisierung der Funktionen  $f_1$  und  $f_2$  gegeben ist. (Sie sollten Disjunktion von Mintermen erhalten).  
 Geben Sie die Kosten  $cost(p_1, p_2) = (cost_1(p_1, p_2), cost_2(p_1, p_2))$  des in Abbildung 1 gegebenen PLA an. (ohne Begründung)
- b) Zeichnen Sie für  $f_1$  und  $f_2$  jeweils einen 4-dimensionalen Würfel (Hypercube), in dem  $ON(f_1)$  bzw.  $ON(f_2)$  markiert ist.



- c) Bestimmen Sie getrennt für  $p_1$  und  $p_2$  mit Hilfe des Quine-McCluskey-Algorithmus' die jeweiligen Primimplikanten. Geben Sie hierbei jeweils für jeden Schritt  $i$  des Algorithmus' die

Mengen  $L_i^M(f_j)$  und  $Prim(f_j)$  an, entweder als Menge von Monomen oder unter Verwendung der in der Vorlesung benutzten abkürzenden Schreibweise (in der Form „01-1“).

- d) Zeichnen Sie die den Primimplikanten von  $f_1$  bzw.  $f_2$  entsprechenden Teilwürfel in den jeweiligen Hypercube ein.



## Ihre Lösung zu Aufgabe 6:

Name: \_\_\_\_\_

Matrikel-Nr.: \_\_\_\_\_

13

Ihre Lösung zu Aufgabe 6 (Fortsetzung):

A large rectangular grid consisting of 20 columns and 25 rows of small squares, intended for handwriting practice or continuation of a solution.

**Aufgabe 7** (8 Punkte)

Zeichnen Sie den 4-Bit-Carry-Ripple-Addierer  $CR_4$  über der Bibliothek  $BIB = \{and_2, or_2, xor_2, not, mux_2\} \cup \{0, 1\}$  mit  $mux_2 : \mathbb{B}^3 \Rightarrow \mathbb{B}$ ,  $mux_2(s, a, b) = \begin{cases} a, & \text{falls } s = 1 \\ b, & \text{falls } s = 0 \end{cases}$ ; verwenden Sie dabei keine hierarchischen Teilschaltkreise.

Kennzeichnen Sie den längsten Pfad in ihrem Schaltkreis und geben Sie dessen Tiefe an.

Bestimmen Sie für jedes Gatter des ermittelten Schaltkreises den Wert des Gatterausgangs für die Belegung

$$b_3 = 1, b_2 = 1, b_1 = 0, b_0 = 1, a_3 = 0, a_2 = 1, a_1 = 0, a_0 = 1, c_{-1} = 1.$$

Ihre Lösung zu Aufgabe 7:

Name: \_\_\_\_\_

Matrikel-Nr.: \_\_\_\_\_

15

Ihre Lösung zu Aufgabe 7 (Fortsetzung):

A large rectangular grid consisting of 20 columns and 25 rows of small squares, intended for handwriting practice or continuation of a solution.

**Aufgabe 8** (8 Punkte)

Konstruieren Sie einen Mealy-Automaten , der eine Binärzahl inkrementiert. Gehen Sie dabei davon aus, dass die Binärzahl mit dem niederwertigsten Bit zuerst gelesen wird (d. h. von rechts nach links). Links vom höchswertigen Bit ist eine Markierung (#) angebracht, die das Ende der Eingabe kennzeichnet. Sie soll nach dem Inkrementieren wieder als letztes Zeichen ausgegeben werden.

Beispiel: Aus #00111 soll #01000 werden.

Zeichnen Sie das Zustandsdiagramm.

Ihre Lösung zu Aufgabe 8:

A large grid of squares, approximately 20 columns by 25 rows, intended for drawing a state transition diagram (Zustandsdiagramm).

Name: \_\_\_\_\_

Matrikel-Nr.: \_\_\_\_\_

17

Ihre Lösung zu Aufgabe 8 (Fortsetzung):

A large rectangular grid consisting of 20 horizontal rows and 25 vertical columns of small squares, intended for handwriting practice.

**Aufgabe 9** (3 + 5 Punkte)

In Abbildung 3 ist das idealisierte Timing-Diagramm für einen Rechner mit verkürzter *Fetch*- und *Execute*-Phase angegeben. Die Kontrolllogik des Rechners wird realisiert wie in Abb. 2 angegeben. Es sollen die Clock-Enable-Signale für die Register I und IN1 erzeugt werden.

Geben Sie die Boolesche Ausdrücke für folgende Signale der Kontrolllogik an:

- Clock-Enable-Signal  $Icken_{pre}$  zum Speichern neuer Befehle im Instruktionssregister  $I$ .
- Clock-Enable-Signal  $IN1cken_{pre}$  zum Speichern neuer Daten im 1. Indexregister  $IN1$ .

Hinweis: Gehen Sie analog zur Vorlesung vor, d.h. es wird eine Kontrolllogik verwendet, wie sie in der Vorlesung vorgestellt wurde. Beachten Sie bei welchen Befehlen des RE-TI Rechners die obigen Signale jeweils aktiv werden. Die Befehlsübersicht incl. Kodierung der Register befindet sich am Ende der Klausur in Tabelle 1 und 2.

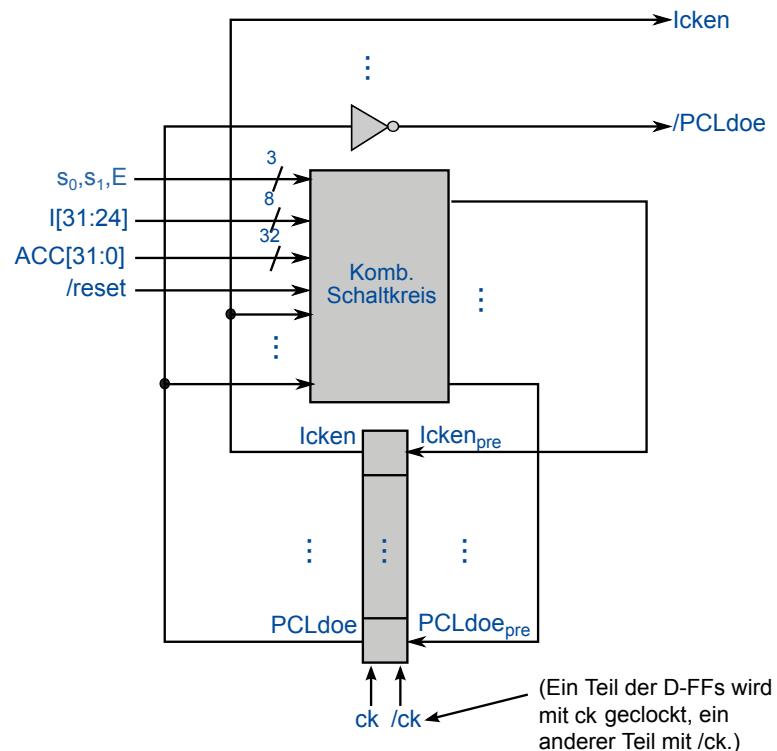


Abbildung 2: Kontrolllogik

**Ihre Lösung zu Aufgabe 9 (Fortsetzung):**

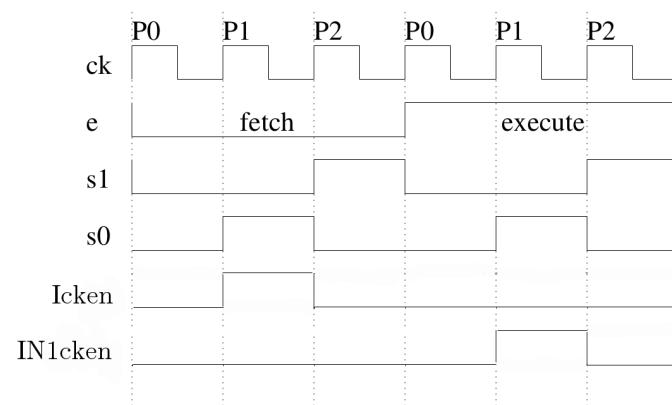


Abbildung 3: Idealisiertes Timing-Diagramm

Ihre Lösung zu Aufgabe 9 (Fortsetzung):

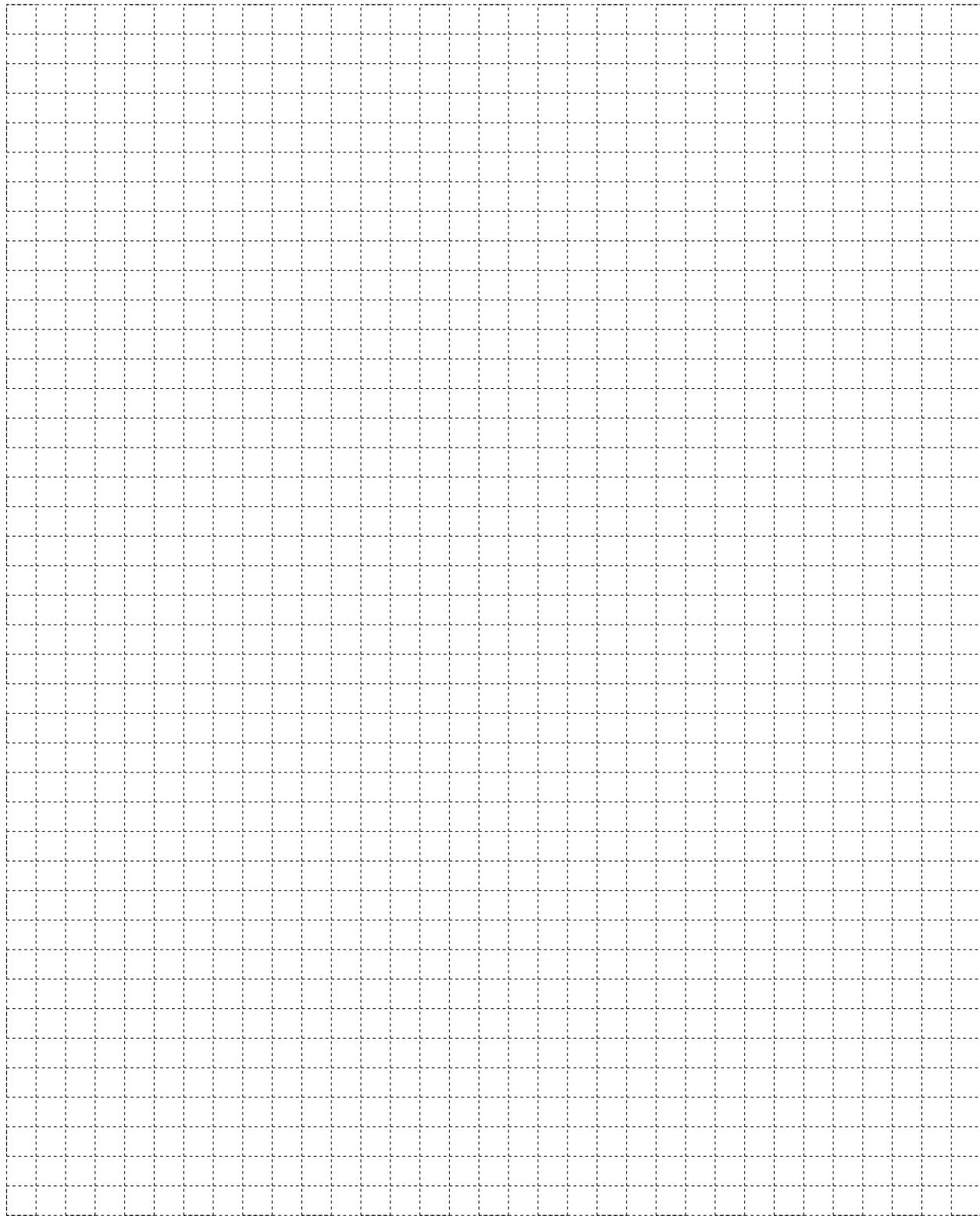
A large rectangular grid consisting of 20 columns and 25 rows of small squares, intended for handwritten solutions.

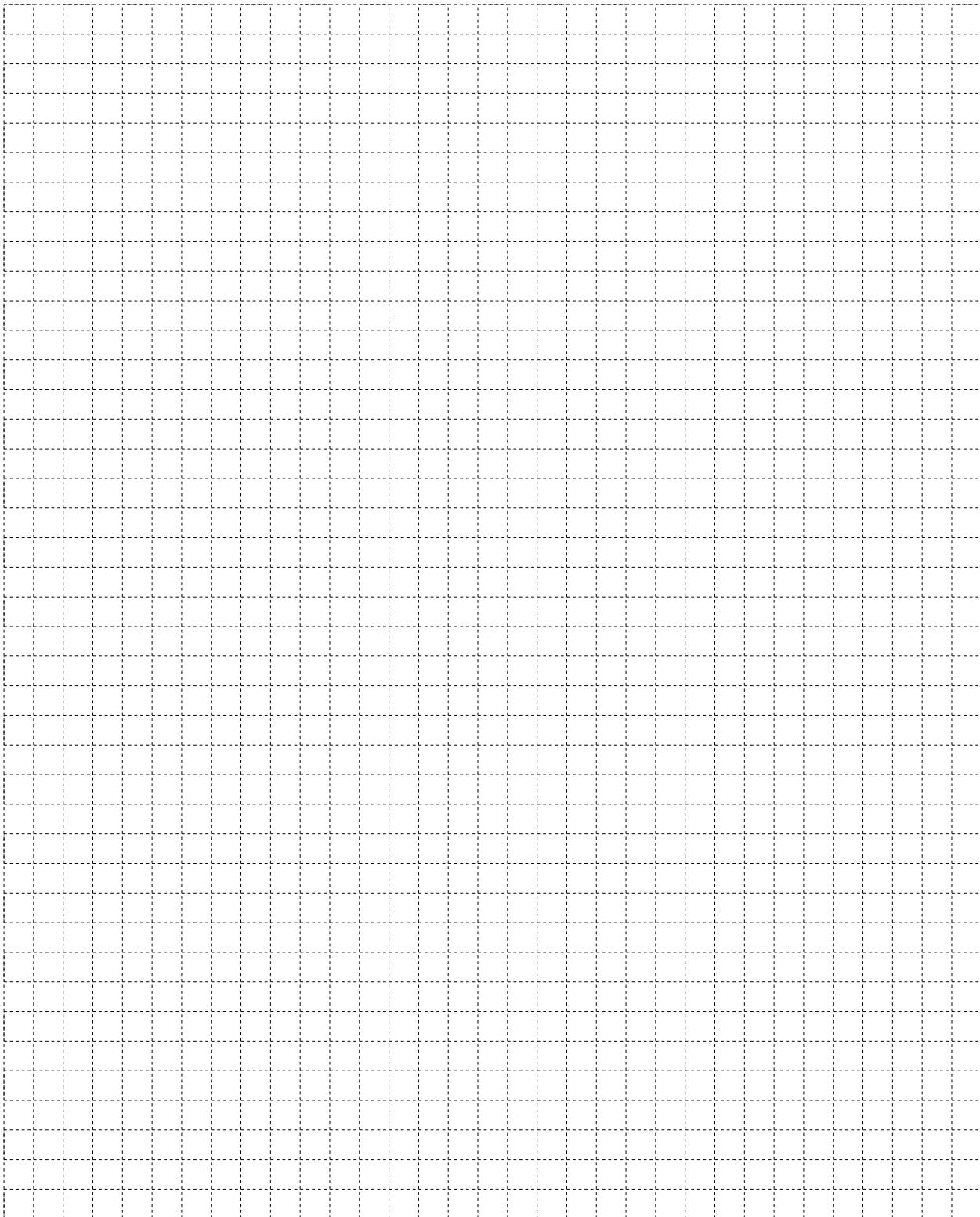
Name: \_\_\_\_\_

Matrikel-Nr.: \_\_\_\_\_

21

**Zusatzseiten für alle Aufgaben – Seite 1**



**Zusatzseiten für alle Aufgaben – Seite 2**

| Load Befehle $I[25, 24] = D$ |               |                                     |
|------------------------------|---------------|-------------------------------------|
| $I[31, 28]$                  | Befehl        | Wirkung                             |
| 0100                         | LOAD $D i$    | $D := M(\langle i \rangle)$         |
| 0101                         | LOADIN1 $D i$ | $D := M(\langle IN1 \rangle + [i])$ |
| 0110                         | LOADIN2 $D i$ | $D := M(\langle IN2 \rangle + [i])$ |
| 0111                         | LOADI $D i$   | $D := 0^8i$                         |

| Store Befehle MOVE: $I[27, 26] = S, I[25, 24] = D$ |              |                                                                                   |
|----------------------------------------------------|--------------|-----------------------------------------------------------------------------------|
| $I[31, 28]$                                        | Befehl       | Wirkung                                                                           |
| 1000                                               | STORE $i$    | $M(\langle i \rangle) := ACC$                                                     |
| 1001                                               | STOREIN1 $i$ | $M(\langle IN1 \rangle + [i]) := ACC$                                             |
| 1010                                               | STOREIN2 $i$ | $M(\langle IN2 \rangle + [i]) := ACC$                                             |
| 1011                                               | MOVE $S D$   | $D := S$ $\langle PC \rangle := \langle PC \rangle + 1, \text{ falls } D \neq PC$ |

| Compute Befehle $I[25, 24] = D$ |              |                                                                                                               |
|---------------------------------|--------------|---------------------------------------------------------------------------------------------------------------|
| $I[31, 26]$                     | Befehl       | Wirkung                                                                                                       |
| 000010                          | SUBI $D i$   | $[D] := [D] - [i]$                                                                                            |
| 000011                          | ADDI $D i$   | $[D] := [D] + [i]$                                                                                            |
| 000100                          | OPLUSI $D i$ | $D := D \oplus 0^8i$ $\langle PC \rangle := \langle PC \rangle + 1, \text{ falls } D \neq PC$                 |
| 000101                          | ORI $D i$    | $D := D \vee 0^8i$                                                                                            |
| 000110                          | ANDI $D i$   | $D := D \wedge 0^8i$                                                                                          |
| 001010                          | SUB $D i$    | $[D] := [D] - [M(\langle i \rangle)]$                                                                         |
| 001011                          | ADD $D i$    | $[D] := [D] + [M(\langle i \rangle)]$                                                                         |
| 001100                          | OPLUS $D i$  | $D := D \oplus M(\langle i \rangle)$ $\langle PC \rangle := \langle PC \rangle + 1, \text{ falls } D \neq PC$ |
| 001101                          | OR $D i$     | $D := D \vee M(\langle i \rangle)$                                                                            |
| 001110                          | AND $D i$    | $D := D \wedge M(\langle i \rangle)$                                                                          |

| Jump Befehle |               |                                                                                                                                                                                      |
|--------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $I[31, 27]$  | Befehl        | Wirkung                                                                                                                                                                              |
| 11000        | NOP           | $\langle PC \rangle := \langle PC \rangle + 1$                                                                                                                                       |
| 11001        | JUMP $> i$    |                                                                                                                                                                                      |
| 11010        | JUMP $= i$    |                                                                                                                                                                                      |
| 11011        | JUMP $\geq i$ |                                                                                                                                                                                      |
| 11100        | JUMP $< i$    | $\langle PC \rangle := \begin{cases} \langle PC \rangle + [i], & \text{falls } [ACC] c 0 \quad (c \in \{<, \leq, =, \geq, >\}) \\ \langle PC \rangle + 1 & \text{sonst} \end{cases}$ |
| 11101        | JUMP $\neq i$ |                                                                                                                                                                                      |
| 11110        | JUMP $< i$    |                                                                                                                                                                                      |
| 11111        | JUMP $i$      | $\langle PC \rangle := \langle PC \rangle + [i]$                                                                                                                                     |

Kodierung der Register: PC 00 / IN1 01 / IN2 10 / ACC 11

Tabelle 1: Befehlstabelle der ReTI

| S, D | Register |
|------|----------|
| 0 0  | PC       |
| 0 1  | IN1      |
| 1 0  | IN2      |
| 1 1  | ACC      |

Tabelle 2: Kodierung S,D von ReTI