

**Vorname:** \_\_\_\_\_

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

**Matr.-Nr.:** \_\_\_\_\_

**Note:** \_\_\_\_\_

27.02.2007, 14.00 Uhr - 16.00 Uhr

**UNIVERSITÄT KARLSRUHE**  
**Institut für Industrielle Informationstechnik**  
- Prof. Dr.-Ing. habil. K. Dostert -

**Vordiplomprüfung im Fach**

***Mikrorechnertechnik***

Die Prüfung umfasst **10 Aufgaben auf 24 Seiten**.

Bitte schreiben Sie auf **alle** Lösungsblätter Ihren Namen und Ihre Matrikelnummer.

Geben Sie bei allen Aufgaben den kompletten Lösungsweg an! Die alleinige Nennung des Endresultats ist zur Erlangung der vollen Punktzahl einer Aufgabe nicht ausreichend.

Die Verwendung eigenen Papiers ist nicht erlaubt.

Bei Bedarf kann zusätzliches Schreibpapier bei der Aufsicht angefordert werden.

Als Hilfsmittel sind Schreib- und Zeichenzeug sowie Taschenrechner mit zu Beginn der Klausur gelöschtem Speicher zugelassen.

Aufgabe:	1	2	3	4	5	6	7	8	9	10	gesamt
Punkte:											
Erreichbare Punktzahl:	12	10	11	10	10	8	10	9	12	8	100

**Aufgabe 1: Speicher und speicherbasierte Anwendungen**

**12 Punkte**

- a) Nennen Sie 4 Typen von Festwertspeichern! Erläutern Sie in Stichworten, ob und gegebenenfalls wie diese Festwertspeicher beschrieben und gelöscht werden können!
- b) Zeichnen Sie eine Speicherzelle eines RAM-Bausteins! Wählen Sie entweder eine SRAM- oder eine DRAM-Zelle. Benennen Sie eindeutig den von Ihnen gewählten Zellentyp und erläutern Sie in Stichworten die Eigenschaften der gewählten Speicherzelle!

**Fortsetzung der 1. Aufgabe:**

Eine spezielle Anwendung von Speichern ist die schnelle speicherbasierte Multiplikation. Im Folgenden werde eine speicherbasierte Multiplikation zweier vorzeichenloser 2 bit-Zahlen betrachtet.

- c) Tabelle 1.1 stelle die jeweils miteinander zu multiplizierenden Faktoren und den Inhalt des Speichers dar. Die obere Zeile sei dabei Faktor 1, die linke Spalte stelle Faktor 2 dar, während die leeren Felder der Tabelle für die Produkte vorgesehen sind. Tragen Sie die entsprechenden Inhalte in Binärdarstellung in die Tabelle ein!

	00	01	10	11
00				
01				
10				
11				

Tabelle 1.1: Faktoren und Speicherinhalt der 2x2 bit Multiplikation

- d) Erläutern Sie anhand der Ergebnisse aus Aufgabenteil c) mit wenigen Sätzen die Funktionsweise der speicherbasierten Multiplikation!

## Aufgabe 2: Zahlendarstellung in Mikrorechnerprogrammen

10 Punkte

Digitale Signalprozessoren verwenden zur Berechnung der verschiedensten Aufgaben die Fraktalzahlendarstellung, welche zunächst betrachtet werden soll.

- a) Welchen minimalen Wert  $a_4$  und welchen maximalen Wert  $b_4$  kann eine 4 bit-Fraktalzahl annehmen?
  
  
  
  
  
  
  
  
  
  
- b) Welchen minimalen Wert  $a_n$  und welchen maximalen Wert  $b_n$  kann eine  $n$  bit-Fraktalzahl annehmen? Geben Sie Ihr Ergebnis in der Form  $a_n \leq x < b_n$  an!
  
  
  
  
  
  
  
  
  
  
- c) Geben Sie die allgemeine Formel zur Darstellung einer  $n$  bit-Fraktalzahl an!
  
  
  
  
  
  
  
  
  
  
- d) Gegeben sei die Dezimalzahl 0,3984375. Geben Sie diese Zahl als 16 bit-Fraktalzahl an!

Stellen Sie die Zahl  $-7,625$  wie folgt dar:

- e) Als Gleitkommazahl gemäß dem IEEE-P754 Single-Precision-Standard!
- f) Als Zweierkomplement-Festkommazahl mit 16 Stellen vor dem Komma und 16 Nachkommastellen!

### Aufgabe 3: Beschreibung einer FSM

11 Punkte

In Tabelle 3.1 sind die Binärkombinationen einer einfachen programmgesteuerten Maschine gegeben. Das Steuerwerk realisiert die aufgeführten Befehle. Als Arbeitsregister stehen ein Akkumulator (A) und ein Eingaberegister (INR) zur Verfügung. Das Signal INIT ist das Hardware-Resetsignal.

Befehl	OP4	OP3	OP2	OP1	Beschreibung
INC A	0	1	1	0	increment A
ANDL A, INR	0	0	1	1	A=(INR) AND (A)
ORL A, INR	1	0	1	0	A=(INR) OR (A)
NOP	0	1	0	0	no operation

Tabelle 3.1: Befehle einer programmgesteuerten Maschine

- a) Vervollständigen Sie den Zustandsgraphen des Steuerwerks in Abbildung 3.1 mit den noch fehlenden Zustandsübergängen! Geben Sie jeweils die OP-Codes in der in Beispiel 3.1 dargestellten Form an!

Beispiel 3.1:  $\overline{OP4}$   $\overline{OP3}$   $\overline{OP2}$   $\overline{OP1}$   $\overline{INIT}$

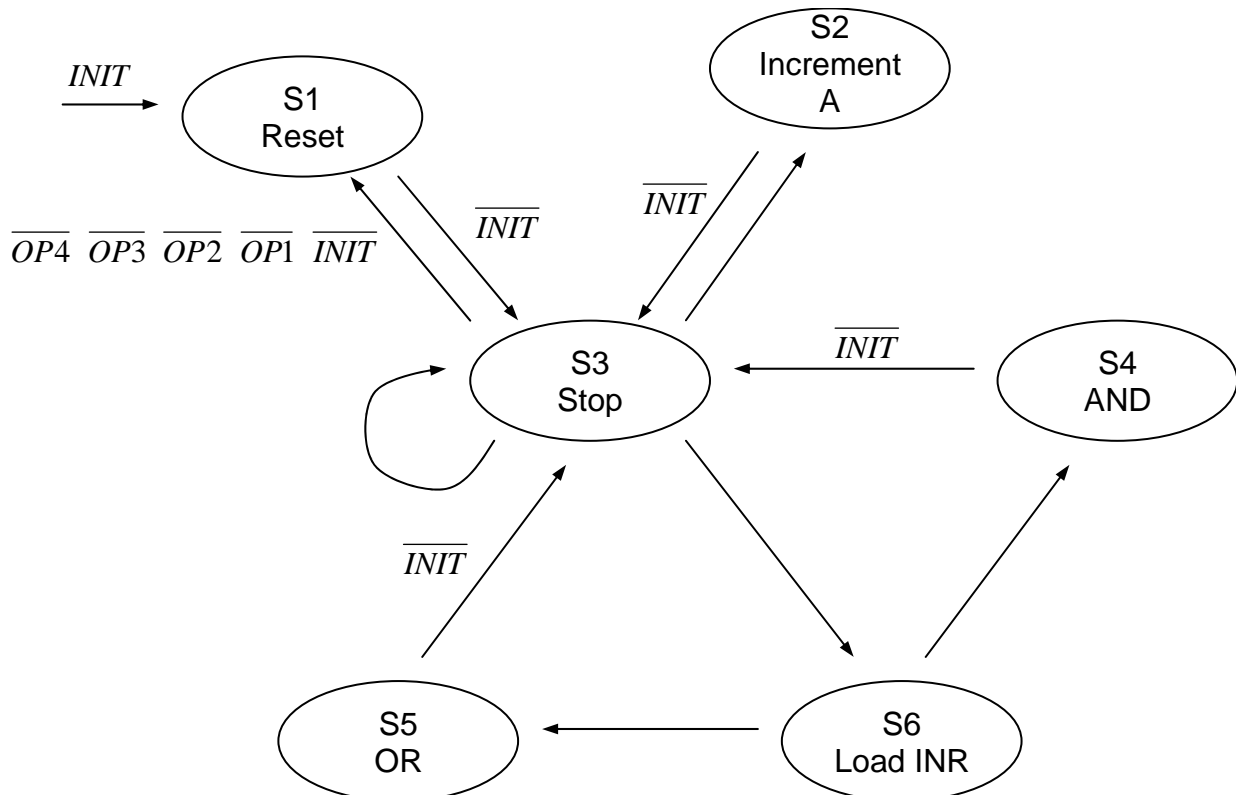


Abbildung 3.1: Zustandsgraph des Steuerwerks

### Fortsetzung der 3. Aufgabe:

In Abbildung 3.2 ist eine Mikrosequenzer-Hardware auf Registertransferebene dargestellt, mit der Werte aus einem Speicher gelesen, in einen Speicher geschrieben und addiert werden können. In der folgenden Tabelle 3.2 ist die Zuordnung einiger Zustände zu den Steuersignalen der Mikrosequenzer-Hardware aufgeführt.

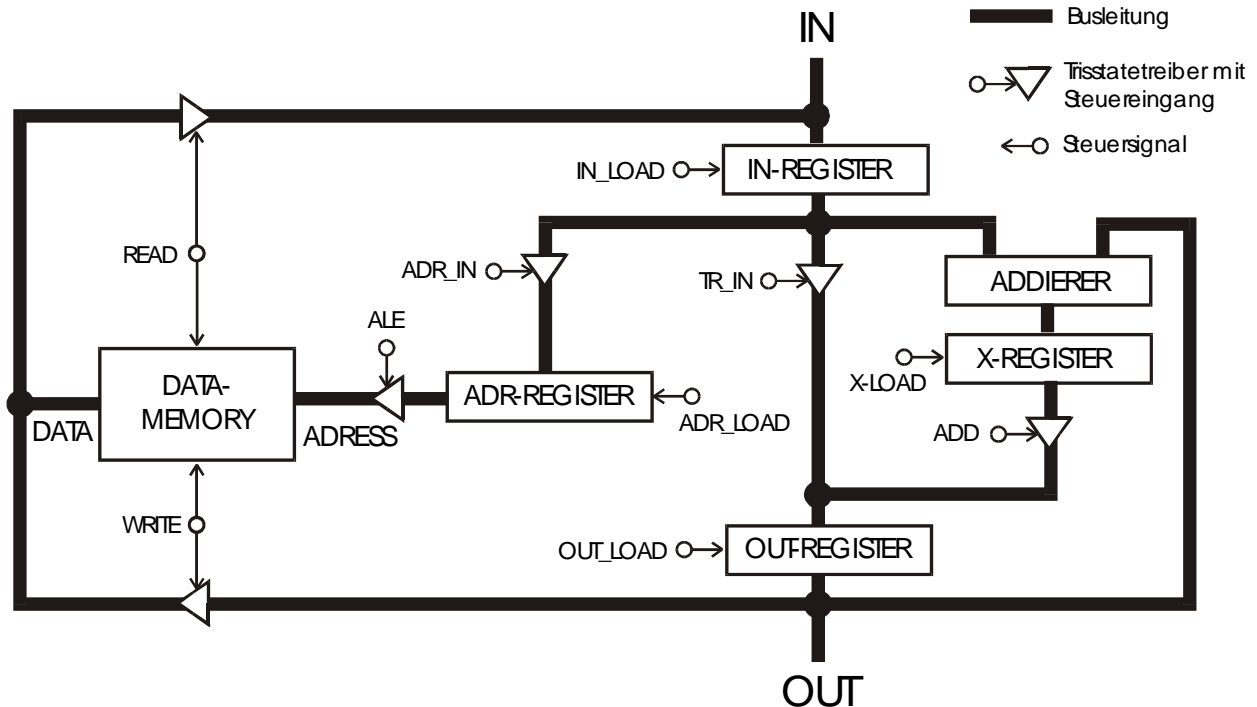


Abbildung 3.2: Mikrosequenzer-Hardware auf Registertransferebene

Zustand	IN_LOAD	ADR_IN	ADR_LOAD	ALE	READ	WRITE	TR_IN	X_LOAD	ADD	OUT_LOAD
S <sub>0</sub>	0	1	1	0	0	0	0	0	0	0
S <sub>1</sub>	0	0	0	0	0	0	0	0	0	0
S <sub>2</sub>	0	0	0	1	0	1	0	0	0	0
S <sub>3</sub>	0	0	0	0	0	0	0	1	0	0
S <sub>4</sub>	1	0	0	0	0	0	0	0	0	0

Tabelle 3.2: Zuordnung der Zustände zu den Steuersignalen

- b) Geben Sie die Folge von Zuständen  $S_i \dots S_j$  an, die notwendig sind, um einen Wert der am Eingang (IN) anliegt, in das ADR-Register zu schreiben (entspricht dem Befehl *MOV ADR, IN*)!

**Fortsetzung der 3. Aufgabe:**

- c) Geben Sie die Folge von Zuständen  $S_i \dots S_j$  an, die notwendig sind, um einen Wert, der im *OUT*-Register liegt, in das *DATA-MEMORY* an die Adresse, die am Eingang *IN* anliegt, zu schreiben!
- d) In der folgenden Tabelle 3.3 sind zwei Kombinationen von Hardware-Steuersignalen vorgegeben. Erläutern Sie in Stichworten, welche beiden Schritte durch die Signalkombinationen ausgeführt werden!

	IN_LOAD	ADR_IN	ADR_LOAD	ALE	READ	WRITE	TR_IN	X_LOAD	ADD	OUT_LOAD
<b>Schritt 1</b>	0	0	0	0	0	0	1	0	0	1
<b>Schritt 2</b>	1	0	0	1	1	0	0	0	1	1



#### Aufgabe 4: CMOS-Transferrates

10 Punkte

In der CMOS-Technologie lassen sich mittels Transferrates komplexe Schaltungen kostengünstig und mit geringem Aufwand realisieren. Im Folgenden soll ein NAND-Gatter betrachtet werden. Abbildung 4.1 zeigt das NAND-Gatter mit den Eingängen  $a$  und  $b$ , sowie dem Ausgang  $c$ .

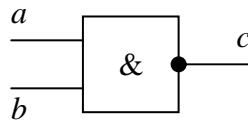


Abbildung 4.1: NAND-Gatter

- a) Vervollständigen Sie die nachfolgende Tabelle 4.1, so dass sich die Funktion eines NAND-Gatters ergibt!

a	b	c

Tabelle 4.1: Wahrheitstabelle

- b) Vervollständigen Sie den Aufbau eines NAND-Gatters in CMOS-Technologie! Verwenden Sie hierzu die in Abbildung 4.2 bereits vorgegebenen n- und p-Kanal-Transistoren und verbinden Sie diese entsprechend mit den Eingängen  $a$  und  $b$  sowie dem Ausgang  $c$ .

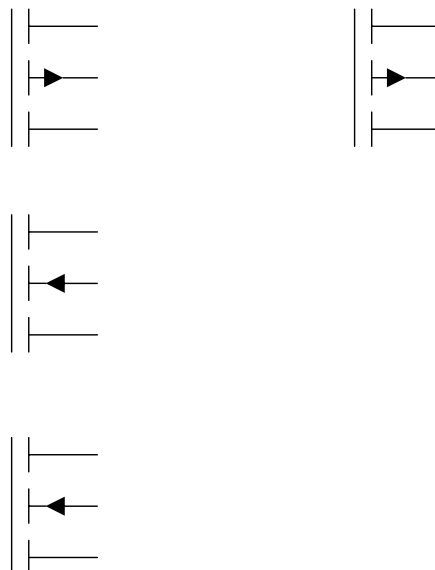


Abbildung 4.2: CMOS-NAND mit einzelnen Transistoren

**Fortsetzung der 4. Aufgabe:**

- c) Skizzieren Sie den Aufbau eines NAND-Gatters mit Transferegates!

### Aufgabe 5: Addierer und Multiplizierer

10 Punkte

- a) Vervollständigen Sie Abbildung 5.1 derart, dass aus den beiden Halbaddierern *HA1* und *HA2* sowie dem ODER-Gatter ein Volladdierer mit den Eingängen *x*, *y* und *z* sowie den Ausgängen *s* (Summe) und *c* (Carry) entsteht!

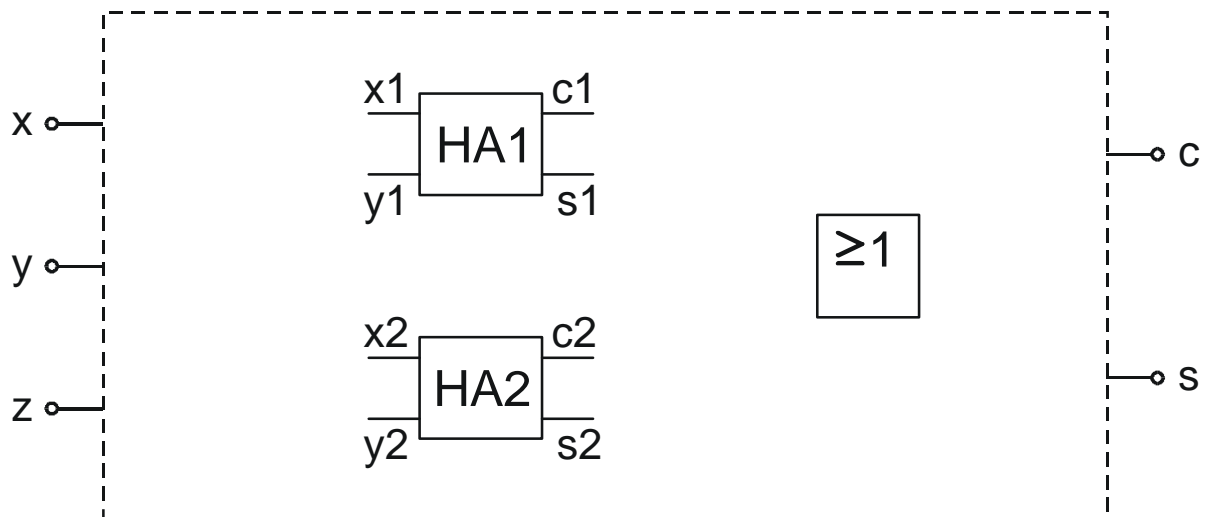


Abbildung 5.1: Aufbau eines Volladdierers

Nun soll ein Carry-Look-Ahead-Addierer aus Volladdierern und einer kombinatorischen Logikschaltung aufgebaut werden. Die kombinatorische Logikschaltung berechnet die Überträge *c0* bis *c3* der Volladdierer VA0 bis VA3 im Voraus.

- b) Ergänzen Sie Abbildung 5.2 zu einem Carry-Look-Ahead-Addierer, der zwei 4 bit-Zahlen *a* und *b* sowie ein Übertragsbit *c* addiert, indem Sie alle fehlenden Verbindungen einzeichnen! Als Ergebnis liefert der Addierer die 5 bit-Summe *s*.

**HINWEIS:** Es werden nicht alle Ausgänge der Volladdierer benötigt.

**Fortsetzung der 5. Aufgabe:**

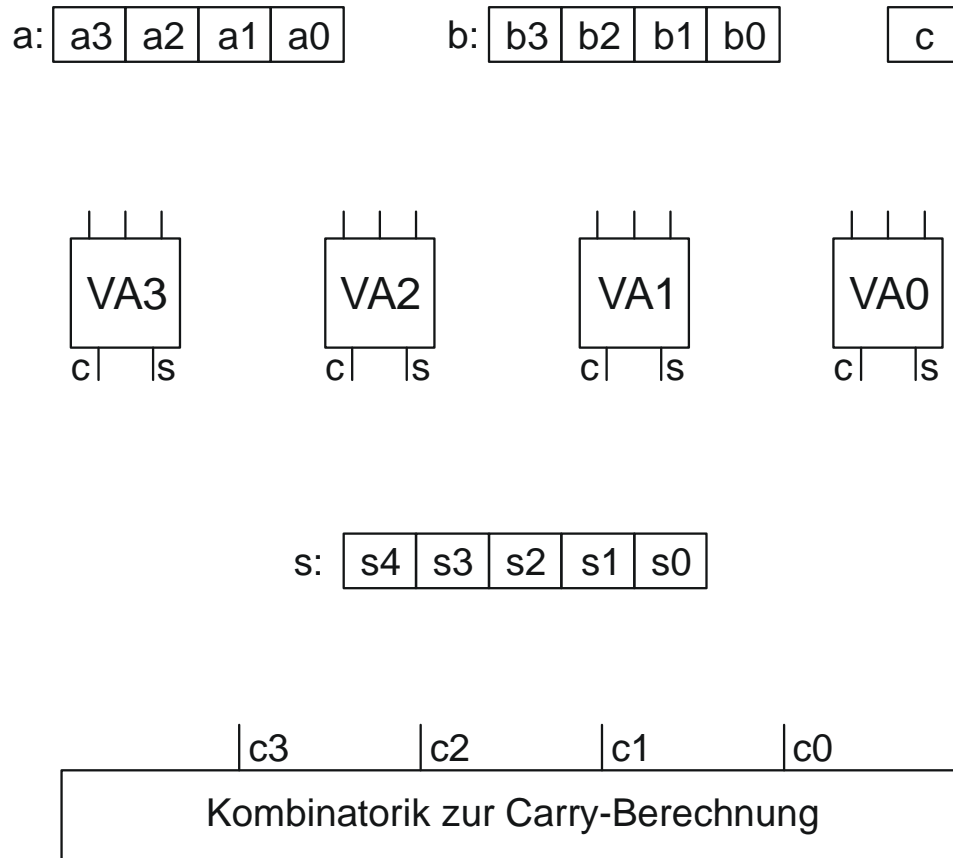


Abbildung 5.2: Carry-Look-Ahead-Addierer zur Addition zweier 4 bit-Zahlen

Im Weiteren wird nun die Arbeitsgeschwindigkeit eines Parallelmultiplizierers für zwei 8-bit-Zweierkomplementzahlen  $A$  und  $B$  betrachtet, der gemäß Gleichung 5.1 ein 16 bit langes Produkt  $P$  als Ergebnis liefert:

$$P = A \cdot B = -2^{15} + 2^8 + a_7 \cdot b_7 \cdot 2^{14} + \sum_{j=0}^6 \sum_{i=0}^6 a_i \cdot b_j \cdot 2^{i+j} + \sum_{i=0}^6 \overline{a_7 b_i} \cdot 2^{i+7} + \sum_{i=0}^6 \overline{b_7 a_i} \cdot 2^{i+7} \quad (5.1)$$

Die Produktbildung der Komponenten  $a_i \cdot b_j$  erfolgt mittels UND-Gattern und die der negierten Komponenten  $\overline{b_i \cdot a_j}$  mittels NAND-Gattern. Die Gatter haben jeweils eine Durchlaufzeit von 1ns.

Die Abarbeitung bis hin zum Endergebnis erfolgt nach dem Schema in Abbildung 5.3, unter Einsatz von Halb- und Volladdierern, sowie eines schnellen Carry-Look-Ahead-Addierers für die Abschlussaddition.

Fortsetzung der 5. Aufgabe:

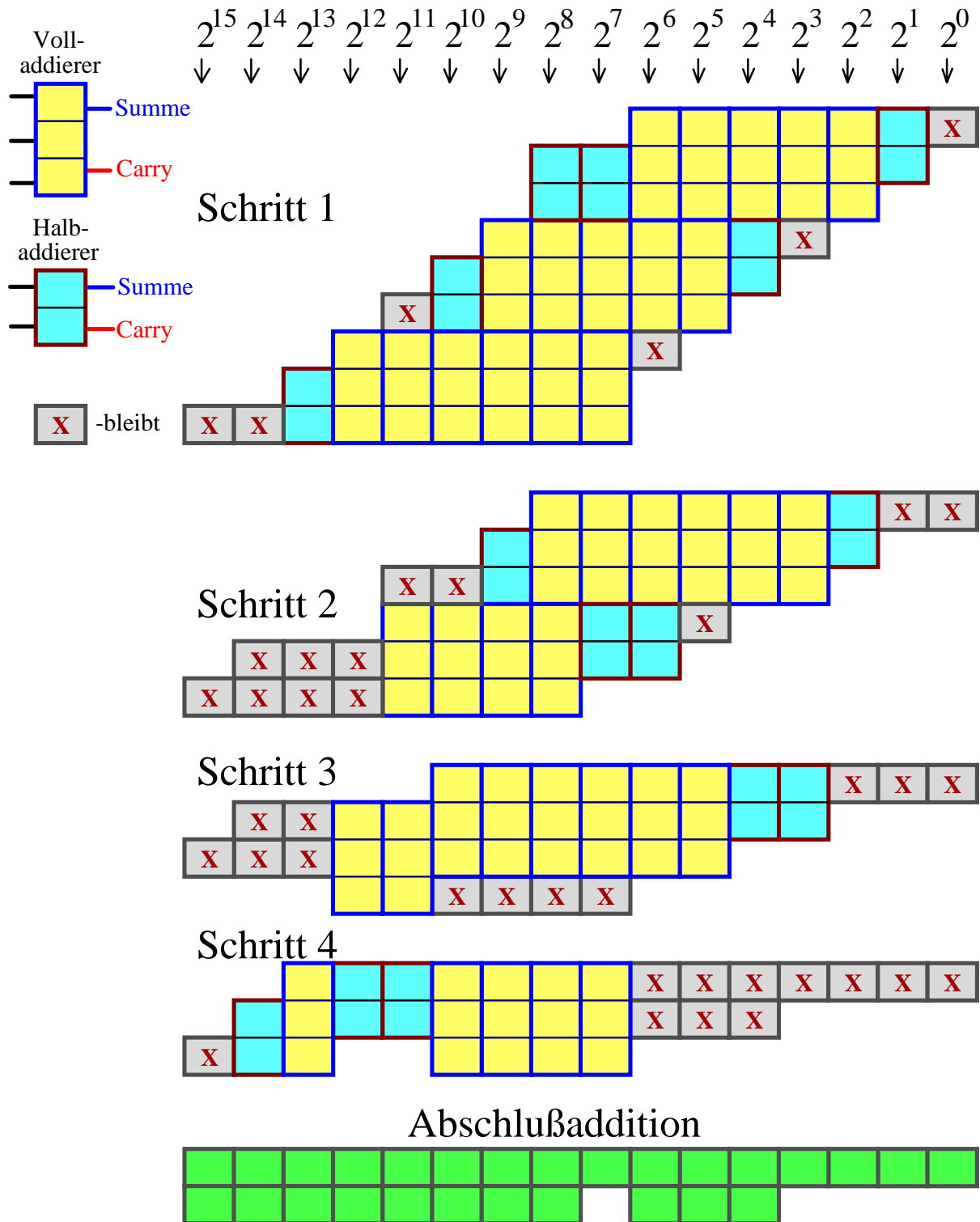


Abbildung 5.3: Schematische Darstellung der Abarbeitung der Multiplikation nach Gleichung 5.1 bis zur Abschlussaddition mit Hilfe von 15 Halb- und 39 Volladdierern in 4 Schritten

**Fortsetzung der 5. Aufgabe:**

- c) Wie lange dauert die Ausführung einer Multiplikation, wenn ein Volladdierer und der Carry-Look-Ahead-Addierer für die Abschlussaddition jeweils 2 ns Durchlaufzeit haben?

Jetzt wird nach jedem der obigen Schritte 1...4 jeweils ein Pipelining-Register eingebaut. Jedes dieser Register weist eine Durchlaufzeit von 0,75 ns auf.

- d) Mit welcher Taktfrequenz  $f_{\max}$  können die Pipelining-Register betrieben werden, damit, nachdem die gesamte Pipeline gefüllt ist, mit jedem Taktschritt ein Multiplikationsergebnis geliefert wird?

- e) Wie viele Bits muss das Register, das nach Schritt 2 einzufügen ist, speichern?

## Aufgabe 6: Analyse eines Mikrocontrollerprogramms

8 Punkte

Es soll ein Programm in 8051-Assemblersprache analysiert werden, das eine arithmetische Operation mit dem Inhalt des Registers *R0* durchführt. Das Register *R0* enthält zu Beginn eine beliebige vorzeichenlose 8 bit Binärzahl. Die Anfangsbelegung der übrigen verwendeten Register bzw. Speicheradressen ist in Tabelle 6.1 angegeben.

Regis- ter/Speicheradresse	R2	R1	20H
Inhalt (hexadezimal)	00	00	00

Tabelle 6.1: Anfangsbelegung der verwendeten Register bzw. Speicherstellen

- a) Analysieren Sie den folgenden Programmcode, indem Sie in jeder Zeile rechts neben dem Befehlscode mit einem kurzen Kommentar angeben, welche Operation durchgeführt wird!

CLR C	
MOV A,R0	
RLC A	
MOV 20H.1,C	
CLR C	
RLC A	
MOV 20H.0,C	
DEC A	
MOV R1,A	
MOV R2,20H	

- b) Welche arithmetische Operation führt das Programm aus?

- c) In welchem Register bzw. in welchen Registern wird das Ergebnis abgelegt?

**Fortsetzung der 6. Aufgabe:**

- d) Tragen Sie in Tabelle 6.2 die Registerbelegung am Ende des Programms ein, wenn *R0* zu Beginn den Wert *C6* (hexadezimal) enthält!

Register	R2	R1	R0
Inhalt (hexadezimal)			

Tabelle 6.2: Registerbelegung nach Ablauf des Programms



## Aufgabe 7: Baudratengenerierung mit dem 80C51

10 Punkte

Unter Verwendung des integrierten Timers 1 bietet der Mikrocontroller 80C51 vielfältige Möglichkeiten der Baudratengenerierung für die serielle Datenübertragung.

Im Folgenden soll die Betriebsart 1 der seriellen Schnittstelle verwendet werden. Diese erlaubt die asynchrone Datenübertragung mit variabler Baudrate (8 bit UART), wobei als Zeitbasis die Überlaufrate von Timer 1 dient.

Zunächst soll eine Baudrate von 4800 bit/s generiert werden. Das Steuerbit SMOD habe den Wert 1. Um die Baudratengenerierung mit möglichst geringer Prozessorbelastung durchzuführen, soll Timer 1 im Autoreload-Modus betrieben werden. Der Autoreload-Wert für Timer 1 sei der Hexadezimalwert *EE*.

- a) Geben Sie die Belegung der Spezialfunktionsregister SCON und TMOD in Tabelle 7.1 an und kennzeichnen Sie irrelevante Bits durch 'x'!

	Bit 7							Bit 0
SCON								
TMOD								

Tabelle 7.1: Belegung der Spezialfunktionsregister

- b) Berechnen Sie die Taktfrequenz  $f_{osc}$ , die am Eingang von Timer 1 benötigt wird, damit eine Baudrate von exakt 4800 bit/s erzeugt wird!

### Fortsetzung der 7. Aufgabe:

- c) Wie groß ist der relative Fehler der erzeugten Baudrate, wenn der Prozessor mit 16 MHz getaktet wird?
- d) Geben Sie für  $SMOD = 1$  die maximale Baudrate bei Betrieb des Timers im Autoreload-Modus an, wenn der Prozessor nun mit 12 MHz getaktet wird!  
(**Hinweis:** Es handelt sich bei der zu berechnenden Baudrate nicht um eine Normbaudrate!)

## Aufgabe 8: Digitale Signalprozessoren

9 Punkte

Mittels eines DSP 56000 soll die Berechnung einer Korrelationsfunktion entsprechend der Gleichung 8.1 durchgeführt werden. Hierzu stehen dem DSP die Daten  $x$  zur Verfügung, die direkt von einem A/D-Wandler eingelesen werden. Die Daten  $y$  seien im Speicher des DSP abgelegt.

$$z = \sum_{k=0}^4 x(k) \cdot y(k). \quad (8.1)$$

Der externe A/D-Wandler ist unter der X-Datenspeicher-Adresse \$1000 angeschlossen und liefert in fünf aufeinander folgenden Maschinenzyklen die folgenden Werte  $x$  in Dezimaldarstellung:

1/32, 1/16, 1/8, 1/4, 1/2

Im Y-Datenspeicher ist ein Ringpuffer der Länge 5 ab Adresse \$40 mit folgendem Inhalt (Daten  $y$ ) angelegt:

Adresse	Inhalt (dezimal)
\$40	1/2
\$41	1/4
\$42	1/8
\$43	1/16
\$44	1/32

Im Folgenden soll schrittweise der Code zur Berechnung der Funktion nach Gleichung 8.1 entwickelt werden.

- Initialisieren Sie die beiden Register  $R0$  und  $R4$  so, dass sie zum Auslesen der Daten aus dem X- und dem Y-Speicher jeweils auf die korrekte Startadresse zeigen!  
(**Hinweis:** Der Zeiger  $R0$  soll für den X-Speicher, der Zeiger  $R4$  für den Zugriff auf den Y-Speicher verwendet werden.)
- Geben Sie für die Berechnung der Korrelationsfunktion ein möglichst kompaktes Assemblerprogramm an!

**Fortsetzung der 8. Aufgabe:**

- c) Tragen Sie in Tabelle 8.1 in Hexadezimalformat die Inhalte in die Teilbereiche des Akkumulators A ein, die sich nach Abarbeitung Ihres Assemblerprogramms gemäß Gleichung 8.1 ergeben!

A2	A1	A0
\$	\$	\$

Tabelle 8.1: Akkumulatorinhalt nach Abarbeitung des Assemblerprogramms gemäß Gleichung 8.1

## Aufgabe 9: Logikbeschreibung mit VHDL

12 Punkte

Im Folgenden soll ein VHDL-Modell schrittweise entwickelt werden. Die Entity soll zwei Eingänge *clk* und *reset* beinhalten, sowie einen Ausgang *result*. Sowohl die Eingänge, als auch der Ausgang seien vom Typ *std\_logic*.

- a) Entwerfen Sie eine Entity mit der Bezeichnung *klausur* mit den entsprechenden Ein- und Ausgängen!  
(**Hinweis:** Gehen Sie davon aus, dass alle benötigten Bibliothekselemente bereits in Ihr Modell eingebunden sind.)

Weiterhin soll nun mit Hilfe eines Prozesses ein Zähler implementiert werden. Der Zähler soll jeweils synchron mit einer steigenden Taktflanke um den Wert 1 erhöht werden. Der Zählvorgang soll mit dem Dezimalwert 1 beginnen und mit dem Dezimalwert 20 enden. Daraufhin soll der Zählvorgang von neuem gestartet werden. Zusätzlich soll der Zähler auf den Wert 1 zurückgesetzt werden, wenn das Eingangssignal *reset* den Wert 0 annimmt, ansonsten soll der Zählvorgang nicht unterbrochen und immer wieder von Neuem gestartet werden. Abbildung 9.1 verdeutlicht diesen Vorgang.

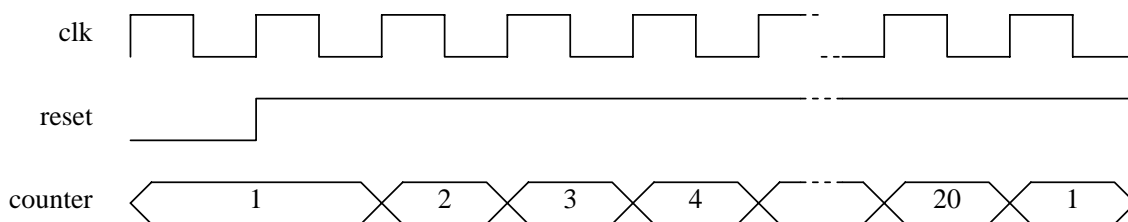


Abbildung 9.1: Schema des Zählvorgangs

- b) Vervollständigen Sie den nachfolgenden Prozess des VHDL-Modells!  
(**Hinweis:** Verwenden Sie zur Implementierung des Zählers ein internes Signal *counter* und gehen Sie davon aus, dass dieses Signal bereits deklariert ist.)

```
process
```

**Fortsetzung der 9. Aufgabe:**

```
end process;
```

Das Signal *counter* wird nun zum Starten eines weiteren Prozesses verwendet. Innerhalb des Prozesses soll der Wert des Signals *counter* abgefragt werden. Nimmt der Zähler Werte aus dem Bereich 1...10 an, so soll das Ausgangssignal *result* einen logischen Low-Pegel annehmen. Für Werte des Zählers im Bereich von 11...20 soll das Ausgangssignal *result* einen logischen High-Pegel annehmen.

- c) Vervollständigen Sie den nachstehenden Prozess so, dass die obigen Bedingungen erfüllt werden!

```
process (counter)
```

```
end process;
```

- d) Welche Funktion wird durch das VHDL-Modell realisiert?

## Aufgabe 10: Analyse von VHDL-Modellen

8 Punkte

VHDL stellt neben Signalen auch Variablen zur Verfügung. Im Folgenden sollen zwei VHDL-Codes auf ihr Zeitverhalten in entsprechenden Simulationen untersucht werden.

- a) Vervollständigen Sie Tabelle 10.1, indem Sie den nachfolgenden VHDL-Code untersuchen und die jeweils resultierenden Werte der Variablen in die Tabelle eintragen!  
(**Hinweis:** Die Variable  $x$  wird extern entsprechend Tabelle 10.1 mit einer Taktrückflanke gesetzt und dem Prozess übergeben!)

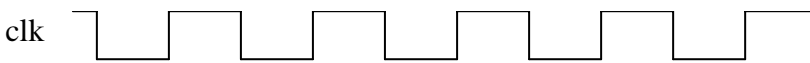
```
process (clk)

variable y1, y2, y3: integer;

begin

    if (clk'event and clk='0') then
        y1 := y2+y3;
        y2 := y3;
        y3 := 5*x;
    end if;

end process;
```



x	3	4	2	1	6
y1					
y2					
y3					

Tabelle 10.1: Timingdiagramm unter Verwendung von Variablen

- b) Vervollständigen Sie Tabelle 10.2, indem Sie den nachfolgenden VHDL-Code untersuchen und die jeweils resultierenden Werte der Signale in die Tabelle eintragen!  
(**Hinweis:** Das Signal  $x$  wird extern entsprechend Tabelle 10.2 mit einer Taktrückflanke gesetzt und dem Prozess übergeben!)

```
signal y1, y2, y3: integer;

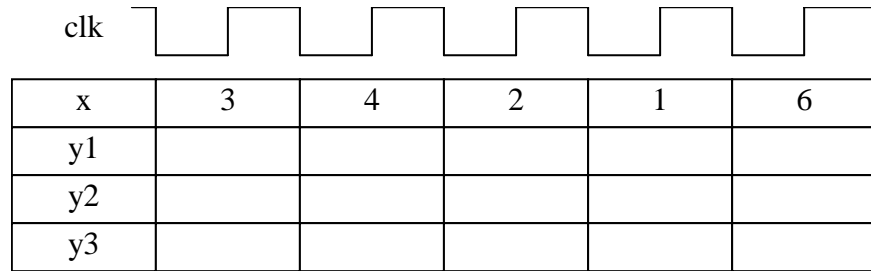
process (clk)

begin

    if (clk'event and clk='0') then
        y1 <= y2+y3;
        y2 <= y3;
        y3 <= 5*x;
    end if;

end process;
```

**Fortsetzung der 10. Aufgabe:**



clk						
x	3	4	2	1	6	
y1						
y2						
y3						

Tabelle 10.2: Timingdiagramm unter Verwendung von Signalen

- c) Nennen Sie drei Objekte, die in VHDL zur Verfügung stehen!
- d) Was wird mit VHDL beschrieben, d.h. wozu wird diese Sprache vorzugsweise eingesetzt? Welche Hauptunterschiede gibt es im Vergleich zu herkömmlichen Programmiersprachen wie C oder Pascal?