Lehrstuhl für Automatisierung und Informationssysteme
Prof. Dr.-Ing. B. Vogel-Heuser

Vorname:	
Nachname:	
Matrikelnummer:	

Prüfung – Informationstechnik

Wintersemester 2016/17

10.03.2017

Bitte legen Sie Ihren Lichtbildausweis bereit.

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält 31 nummerierte Seiten inkl. Deckblatt.

Bitte prüfen Sie die Vollständigkeit Ihres Exemplars!

Bitte nicht mit rot oder grün schreibenden Stiften oder Bleistift ausfüllen!

Diesen Teil nicht ausfüllen.

Aufgabe	GL	BS	MSE	C		∑	**Note**
erreichte Punkte							
erzielbare Punkte	48	48	48	96		240	

Aufgabe G: Grundlagen

Aufgabe G:
48 Punkte

1. Umrechnung zwischen Zahlensystemen

Überführen Sie die unten angegebenen Zahlen in die jeweils anderen Zahlensysteme.
Wichtig: Achten Sie genau auf die jeweils angegebene Basis!

1 (121)$_3$ = ( )$_{10}$ = ( )$_6$

2 (10,375)$_{10}$ = ( )$_2$

2. IEEE 754 Gleitkommazahlen

Rechnen Sie die gegebene Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) in eine Dezimalzahl um.

0	1	0	0	1	0	0	0	1	1
V	e (4 Bit)				M (5 Bit)				

Hinweis: Ergebnisse und Nebenrechnungen außerhalb der dafür vorgesehenen Textblöcke werden nicht bewertet!

Vorzeichen

V=

Bias und biased Exponent

B= **e =**

Exponent

E =

Mantisse (Dualzahl und Denormalisiert)

M_2=

Vollständige Dezimalzahl Z (inkl. Vorzeichen)

Z=

3. Logische Schaltungen und Schaltbilder

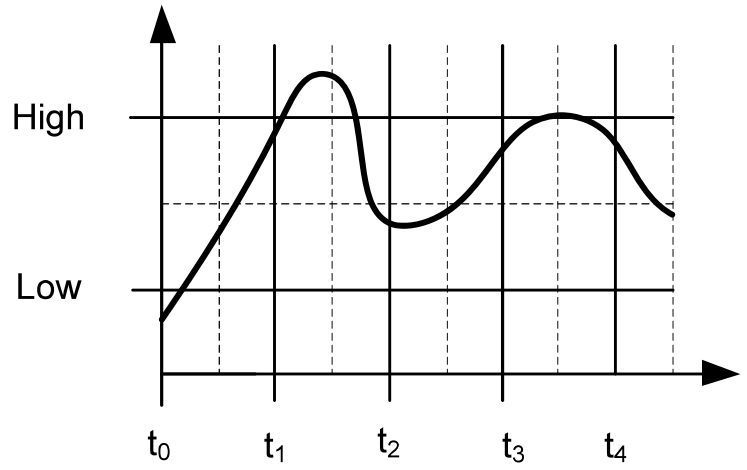
Sie sind zuständig für den Schaltungsentwurf. Ihnen wurde die angegebene Wahrheitstabelle übergeben. Erstellen Sie eine *graphische Schaltung in Normalform* (KNF / DNF). Erstellen sie diejenige Normalform, die am *wenigsten Schaltglieder* erfordert.

a	b	c	y_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

4. Diskretisierung von Signalen

Das folgende Analogsignal soll als Eingangssignal für ein FlipFlop dienen. Hierfür muss das zeit- und wertkontinuierliche Signal diskretisiert werden. Diskretisieren Sie das gegebene Signal *wertdiskret und zeitkontinuierlich* zwischen den Zeitpunkten $t_1$bis einschließlich t_4.

Hinweis: Die Diskretisierungsschwelle ist immer in der Mitte „High“ und „Low“.



5. Normalformen und Minimierung

Gegeben sind folgende KV-Diagramme.

☐ **Dieses KV-Diagramm werten**

	a		**ā**	
b	X	0	0	1
b̄	X	1	1	X
	c̄	**c**		**c̄**

Bild G-5.1: *KV-Diagramm 1*

☐ **Dieses KV-Diagramm werten**

	a		**ā**	
b	X	0	0	1
b̄	X	1	1	X
	c̄	**c**		**c̄**

Bild G-5.2: *KV-Diagramm 2*

a) Minimieren Sie das KV-Diagramm in Form der DNF *(Disjunktive Normalform)* durch Einrahmen (Schleifen) der entsprechenden Felder im oben dargestellten KV-Diagramm. Schreiben Sie die minimierte Funktion in boolescher Algebra in das Lösungsfeld unten auf. Die Felder mit y=„X“ sind don‘t care bits.

Hinweis: Das zweite abgebildete KV-Diagramm dient als Ersatz, falls Sie sich verzeichnen. Kennzeichnen Sie durch Ankreuzen im Feld „dieses KV-Diagramm werten", welches KV-Diagramm bewertet werden soll.

b) Übertragen Sie die gegebene Wahrheitstabelle in eine Konjunktive Normalform (KNF) der booleschen Algebra und minimieren Sie die Schaltung mithilfe von Rechenregeln der booleschen Algebra.

Hinweis: Schreiben Sie alle Zwischenschritte in das Lösungsfeld!

a	**b**	**y_1**
0	0	1
0	1	1
1	0	0
1	1	0

6. Flip-Flops

Gegeben ist die folgende Master-Slave Flip-Flop Schaltung (MS-FF)

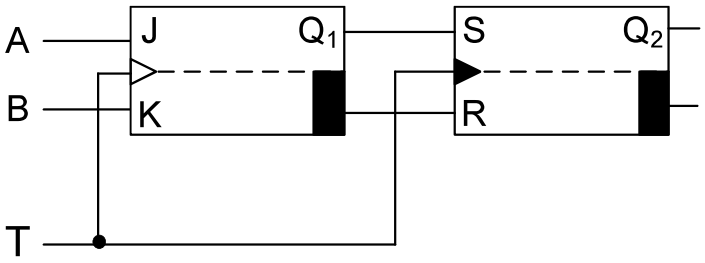
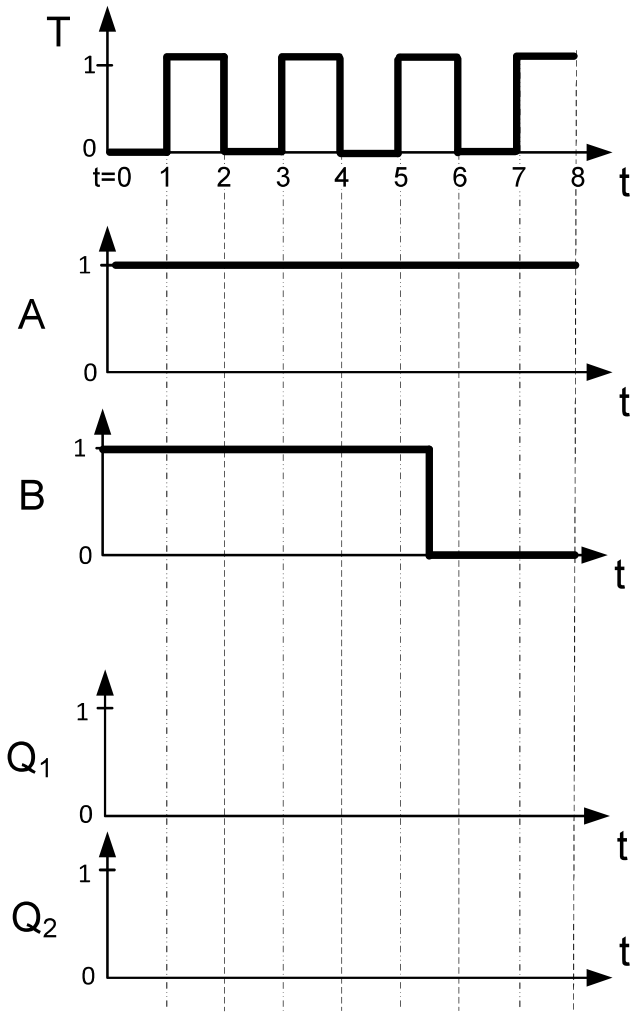


Bild G-6.1: *MS-FF*

Bei t = 0 sind die Flip-Flops in folgendem Zustand: $Q_1 = Q_2 = 1$.

Analysieren Sie die gegebene Flip-Flop Schaltung, indem Sie für die Eingangssignale A, B und T die zeitlichen Verläufe für Q_1 und Q_2 in die vorgegebenen Koordinatensysteme im Lösungsfeld eintragen. Die gegebenen Eingangssignale A, B und T sind in der Abbildung des Lösungsfeldes gegeben.

Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden.



7. MMIX-Rechner

Im Registerspeicher eines MMIX-Rechners (Bild G-7.1) befinden sich die in Bild G-7.2 gegebenen Werte. Es sollen nacheinander die zwei Befehle (Bild G-7.4) abgearbeitet und das Ergebnis in dem Registerspeicher (Bild G-7.2) bzw. Datenspeicher (Bild G-7.3) abgelegt werden.

	0x_0	0x_1	…	0x_4	0x_5	…
	0x_8	0x_9		0x_C	0x_D	
0x0_	TRAP	FCMP	…	FADD	FIX	…
	FLOT	FLOT I		SFLOT	SFLOT I	
0x1_	FMUL	FCMPE	…	FDIV	FSQRT	…
	MUL	MUL I		DIV	DIV I	
0x2_	ADD	ADD I	…	SUB	SUB I	…
	2ADDU	2ADDU I		8ADDU	8ADDU I	
…	…	…	…	…	…	…
0x8_	LDB	LDB I	…	LDW	LDW I	…
	LDT	LDT I		LDO	LDO I	
0x9_	LDSF	LDSF I	…	CSWAP	CSWAP I	…
	LDVTS	LDVTS I		PREGO	PREGO I	
0xA_	STB	STB I	…	STW	STW I	…
	STT	STT I		STO	STO I	
…	…	…	…	…	…	…
0xE_	SETH	SETMH	…	INCH	INCMH	…
	ORH	ORMH		ANDNH	ANDNMH	
0xF_	JMP	JMP B	…	GETA	GETA B	…

Bild G-7.1: *MMIX-Code-Tabelle*

Registerspeicher	
Adresse	Wert vor Befehlsausführung
…	…
$0x87	0x00 00 00 00 00 01 B0 0F
$0x88	0x00 00 00 00 DE AC 10 CF
$0x89	0x00 00 00 00 00 00 00 00
$0x8A	0x00 00 00 00 00 00 61 FF
…	…

Bild G-7.2: *Registerspeicher*

Datenspeicher	
Adresse	Wert
…	…
0x00 00 00 00 00 00 61 FF	0xF0
0x00 00 00 00 00 00 62 00	0x01
0x00 00 00 00 00 00 62 01	0xDA
0x00 00 00 00 00 00 62 02	0x53
0x00 00 00 00 00 00 62 03	0x1B
0x00 00 00 00 00 00 62 04	0x00
0x00 00 00 00 00 00 62 05	0xB0
…	…

Bild G-7.3: *Datenspeicher*

Nr.	Maschinensprache	Assemblersprache	Befehlsbeschreibung
1	0x20 89 87 88	?	?
2	?	?	$0x89=$M_2$[$0x8A+0x02]

Bild G-7.4: *Befehle in Maschinensprache, Assemblersprache und Befehlsbeschreibung*

Bearbeiten Sie nun folgende Fragen (nächste Seite) zu den Befehlen und zu Änderungen, die sich durch die Befehle ergeben.

Befehl Zeile 1:

7.1 Geben Sie für den in Nr. 1 in Bild G-7.4 angegebenen Befehl an, wie dieser in Assemblersprache formuliert ist.

7.2 Wie lautet die Befehlsbeschreibung des in Nr. 1 in Bild G-7.4 angegebenen Befehls? *Hinweis:* Ein Beispiel für eine Befehlsbeschreibung ist in Bild G-7.4, Zeile Nr. 2.

7.3 Geben Sie die Adresse und den neuen Wert der durch den Befehl Nr. 1 in Bild G-7.4 geänderten Registerspeicherzelle an!

Befehl Zeile 2:

7.4 Geben Sie für den in Nr. 2 in Bild G-7.4 angegebenen Befehl an, wie dieser in Maschinensprache formuliert ist. *Hinweis:* Ein Beispiel für einen Befehl in Maschinensprache ist in Bild G-7.4, Zeile Nr. 1.

7.5 Geben Sie die Bytelänge des Zugriffs des Befehls und den Wert der durch Befehl Nr. 2 in Bild G-7.4 geänderten Register- oder Datenspeicherzelle an!

Allgemein zu MMIX:

7.6 Mit welcher „Endianness“ arbeitet MMIX? Was bedeutet das für das Speichern von aus mehreren Bytes zusammengesetzten Worten?

Aufgabe BS: Betriebssysteme

Aufgabe BS:
48 Punkte

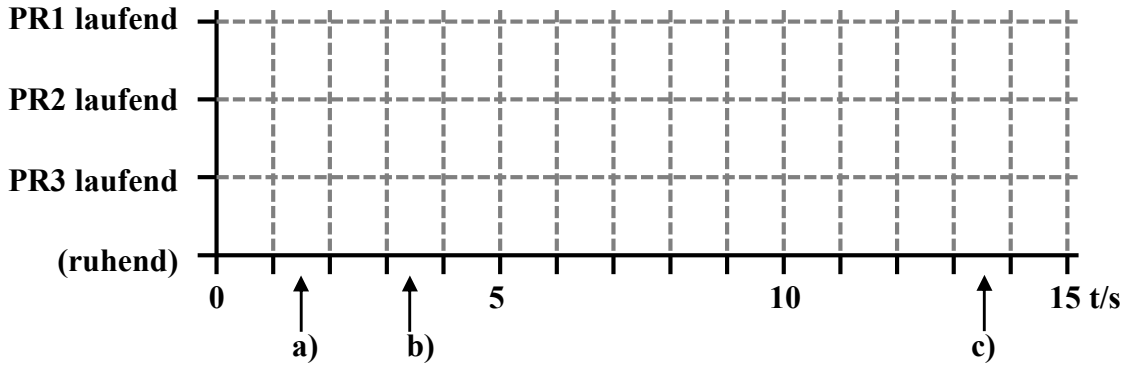
8. Asynchrones Scheduling, nicht-präemptiv

Gegeben seien die folgenden drei Prozesse (Bild BS-8.1) mit den Prioritäten niedrig bis hoch, welche jeweils ab dem Zeitpunkt „Start“ mit der Häufigkeit „Periode“ periodisch aufgerufen werden sollen. Zur Abarbeitung eines Tasks wird die Zeitspanne „Dauer“ benötigt. Erstellen Sie im untenstehenden Diagramm das asynchrone, nicht-präemptive Scheduling der Tasks 1 bis 3 für den Zeitraum 0 bis 15 s für einen Einkernprozessor. Kreuzen Sie danach an den durch einen Pfeil markierten Stellen den aktiven Prozess an.

Hinweis: Nur Antworten innerhalb der Lösungskästen werden gewertet!

Task	Priorität	Start	Periode	Dauer
Pr1	Hoch	7 s	20 s	2 s
Pr2	Mittel	0 s	4 s	1 s
Pr3	Niedrig	0 s	2 s	1 s

Bild BS-8.1: *Taskspezifikation I*



a) PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

b) PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

c) PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

Zu welchem Zeitpunkt wird die Forderung nach Rechtzeitigkeit verletzt?

s

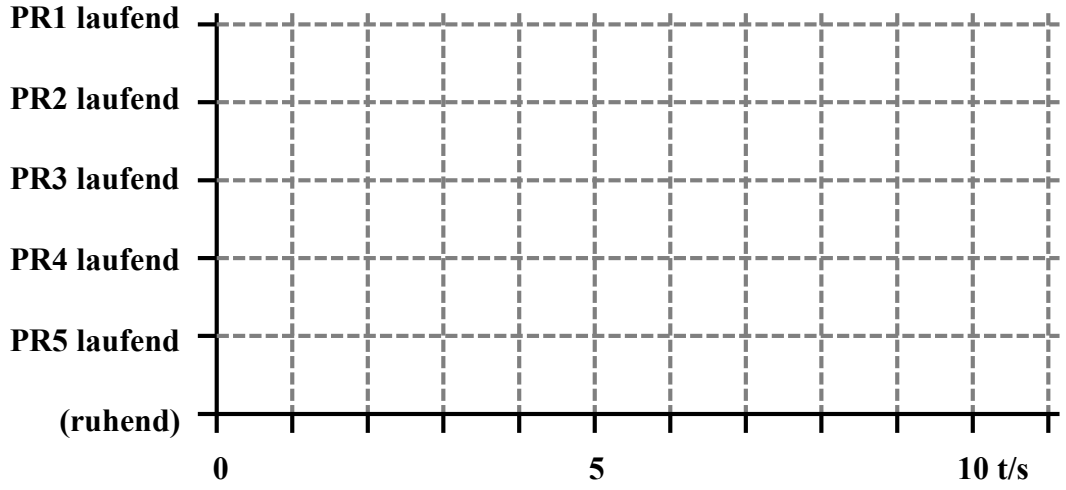
9. Asynchrones Scheduling, präemptiv

Gegeben seien die folgenden fünf Prozesse (Bild BS-9.1), welche jeweils ab dem Zeitpunkt „Start“ eingeplant werden sollen. Zur Abarbeitung eines Tasks wird die Rechenzeitspanne „Dauer“ benötigt. Erstellen Sie im untenstehenden Diagramm das präemptive Scheduling nach dem Schema „Round-Robin“ unter Berücksichtigung der „Prioritäten“ der Tasks 1 bis 5 für den Zeitraum 0 bis 11 s für einen Einkernprozessor. Ein Zeitschlitz hat eine Größe von einer Sekunde. Tragen Sie anschließend die Zeitpunkte, an denen die Tasks fertig abgearbeitet werden, mit einer Genauigkeit von 0,5 s in den untenstehenden Lösungskasten ein.

Hinweis: Nur Antworten innerhalb des Lösungskastens werden gewertet!

Task	Priorität	Start	Dauer
Pr1	Hoch	5 s	1,5 s
Pr2	Mittel	1 s	2 s
Pr3	Mittel	1,5 s	0,5 s
Pr4	Niedrig	0 s	2 s
Pr5	Niedrig	5 s	3 s

Bild BS-9.1: *Taskspezifikation II*



PR1:

PR2:

PR3:

PR4:

PR5:

10. Asynchrone Programmierung

Die zwei periodischen Prozesse PR2 und PR3 sowie der aperiodische Prozess PR1 (Bild BS-10.1) sollen mit dem Verfahren der asynchronen Programmierung präemptiv auf einem Einkernprozessor eingeplant werden. Der Prozess PR1 besitzt die höchste, der Prozess PR3 die niedrigste Priorität. Die Ausführung wird durch einen Interrupt unterbrochen.

Tragen Sie in das unten angegebene leere Diagramm den Verlauf der Abarbeitung von Prozessen und Interrupts ein. Kreuzen Sie danach an den durch einen Pfeil markierten Stellen den aktiven Prozess an.

Hinweis: Nur Kreuze innerhalb des Lösungskastens werden gewertet!

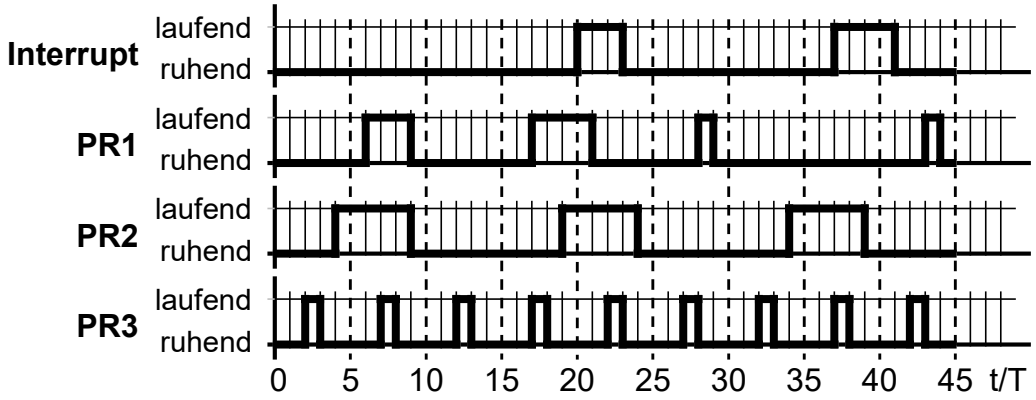
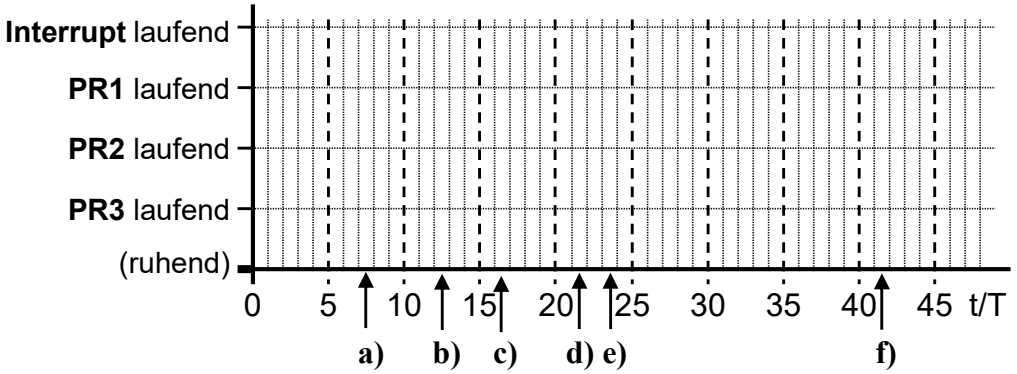


Bild BS-10.1: *Prozessverlauf*



a) Interrupt ( ), PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

b) Interrupt ( ), PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

c) Interrupt ( ), PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

d) Interrupt ( ), PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

e) Interrupt ( ), PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

f) Interrupt ( ), PR1 ( ), PR2 ( ), PR3 ( ), ruhend ( )

11. Semaphoren

Gegeben seien die folgenden vier Tasks T1 bis T4 mit absteigender Priorität sowie die dazugehörigen Semaphoren S1 bis S4 (Bild BS-11.1). Die Startwerte der Semaphoren entnehmen Sie der Antworttabelle. Tragen Sie in der ersten Spalte der Antworttabelle den aktuell laufenden Task ein sowie im Rest der Zeile die Werte der Semaphoren nach Ausführung des jeweiligen Tasks.

T1	T2	T3	T4
P(S1)	P(S2)	P(S3)	P(S4)
	P(S2)	P(S3)	
…	…	…	…
		V(S4)	
V(S3)	V(S1)	V(S4)	V(S3)

Bild BS-11.1: *Semaphorenzuweisung*

Task	S1	S2	S3	S4
-	0	3	1	0

Beantworten Sie außerdem, ob im betrachteten Schedule folgende Phänomene auftreten:

a) Partielle Verklemmung: ( ) ja ( ) nein

b) Globale Verklemmung: ( ) ja ( ) nein

c) Endlosschleife: ( ) ja ( ) nein

12. Echtzeitbetriebssysteme

Im Folgenden (Bild BS-12.1) ist ein unvollständiges „erweitertes Taskzustandsdiagramm von RTOS-UH“ gegeben.

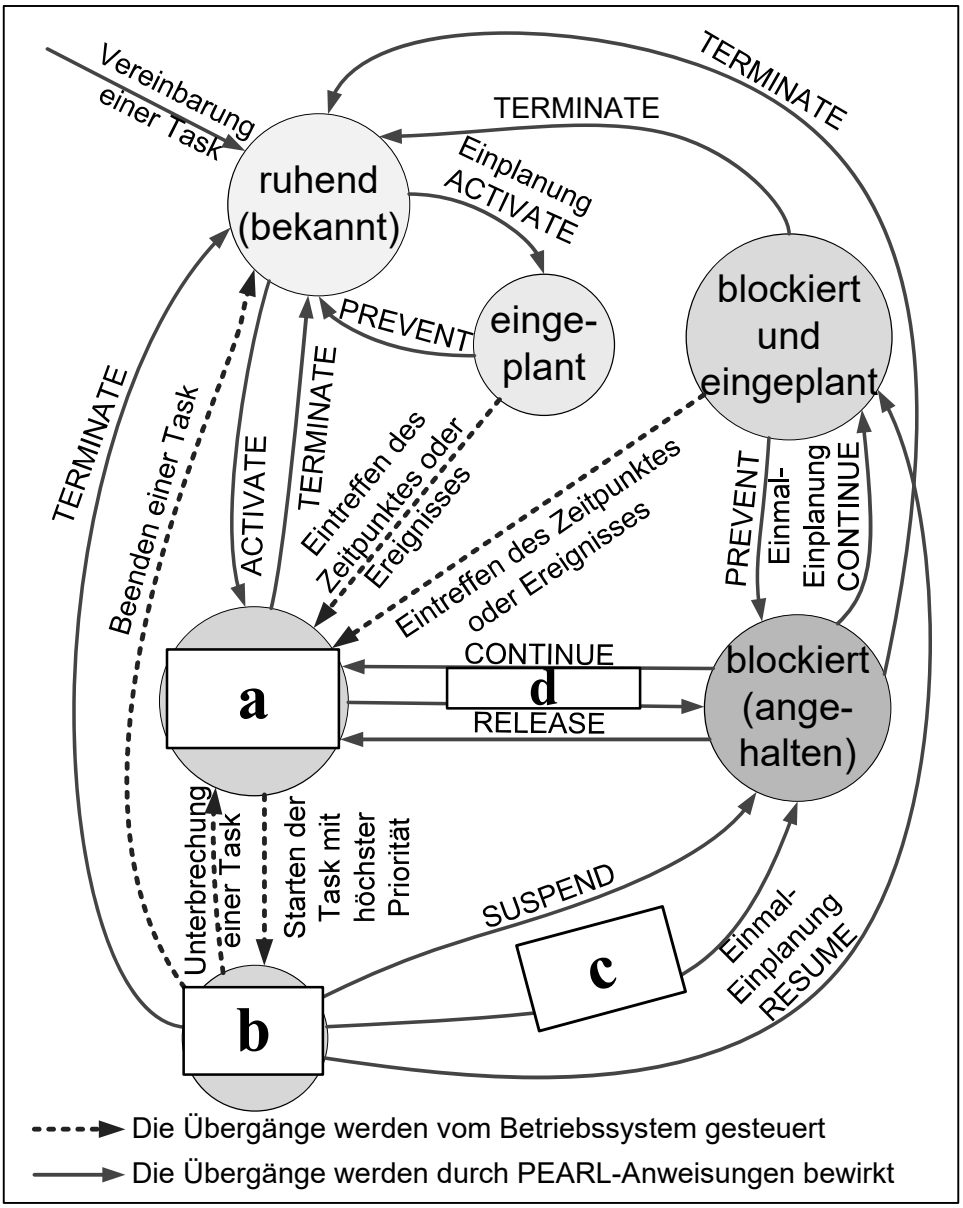


Bild BS-12.1: *Erweitertes Taskzustandsdiagramm von RTOS-UH*

Bezeichnen Sie die im Diagramm mit Buchstaben markierten Lücken:

a) _____

b) _____

c) _____

d) _____

13. IEC 61131-3: Funktionsbausteinsprache (FBS)

Ein Kleingüteraufzug („Essensaufzug“), siehe Bild BS-13.1, verbindet in einem Restaurant eine Küche im Keller (unten) mit der Essensausgabe im Erdgeschoss (oben) und soll mit Hilfe der Funktionsbausteinsprache programmiert werden. Verwenden Sie dafür die untenstehende Beschreibung und Vorlage!

Der Aufzug hat zwei Endpositionen, an denen jeweils ein Endlagensensor („Aufzug_oben“ und „Aufzug_unten“, jeweils wahr, wenn der Aufzug sich an der Position befindet), sowie je einen Knopf („Knopf_oben“ und „Knopf_unten“, jeweils wahr wenn gedrückt) zur Steuerung des Aufzuges sind. Zusätzlich verfügt der Aufzugkasten über eine Klappe mit Sensor („Klappe_zu“, wahr wenn Klappe zu ist), die die Sicherheit des Systems verbessert.

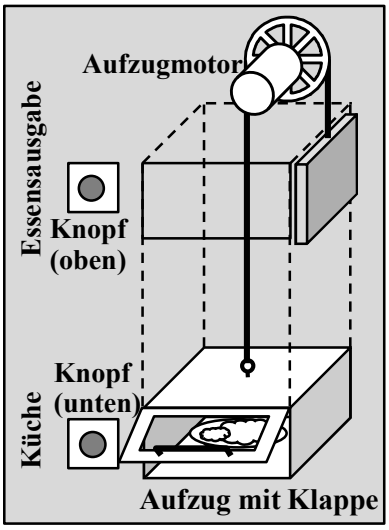
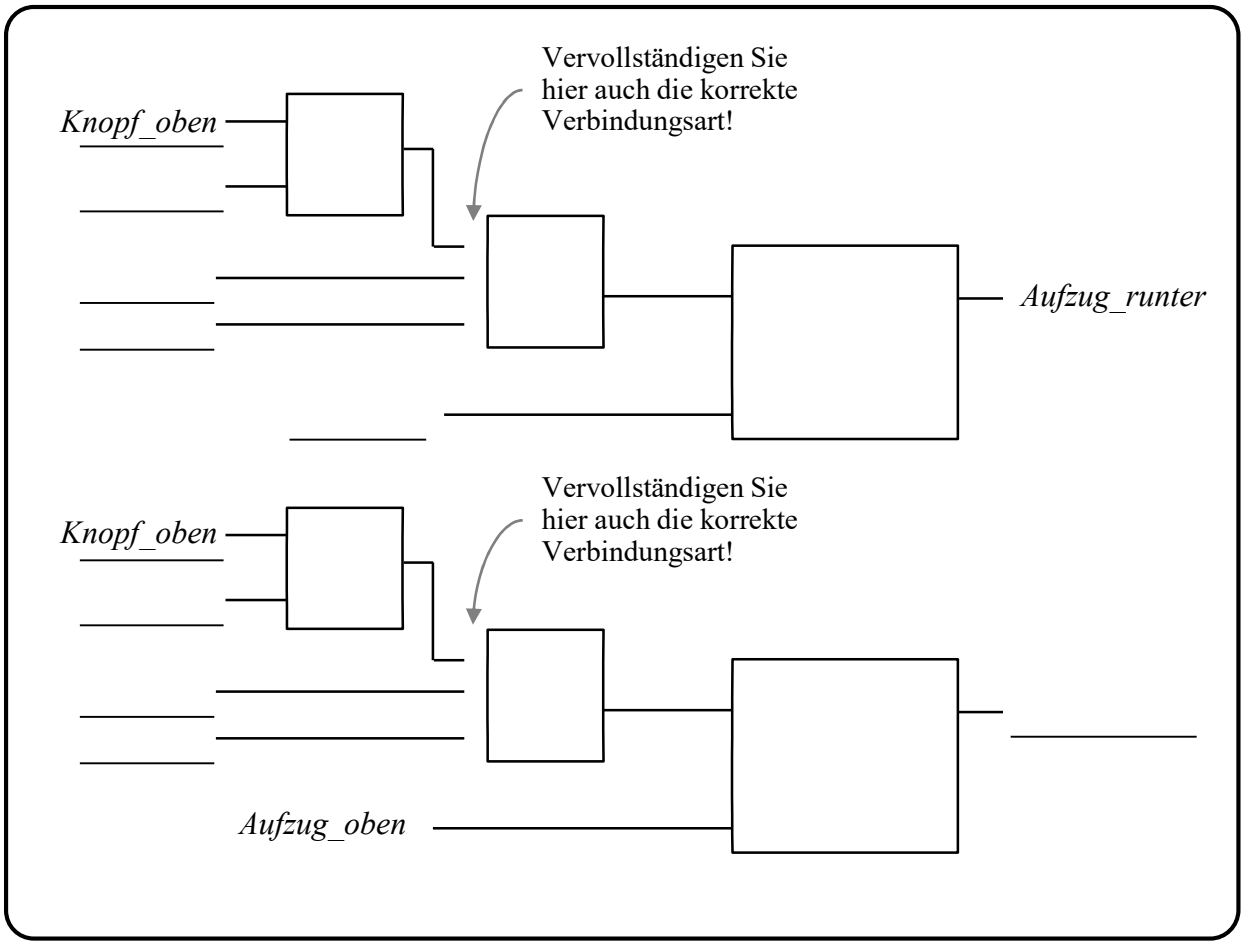


Bild BS-13.1

Im Gegensatz zu einem normalen Aufzug kann dieser Aufzug per Knopfdruck gerufen, aber auch geschickt werden: Befindet sich der Aufzug an einer der Endpositionen (oben oder unten), wird durch Drücken des oberen oder unteren Knopfes der Aufzug an die andere Endposition verfahren, vorausgesetzt die Klappe ist zum Zeitpunkt des Knopfdruckes geschlossen. Durch passendes Schalten des Aufzugmotors („Aufzug_hoch“ und „Aufzug_runter“, ist entweder der eine oder andere Wert wahr, fährt der Aufzug in die „wahre“ Richtung) wird der Aufzug daraufhin verfahren. Wird eine Endposition erreicht wird das jeweilige Motorsignal zurückgesetzt. Achten Sie darauf, dass bei verklemmtem Knopf der Aufzug nicht ständig hin- und herfährt!



Aufgabe MSE: Modellierung und Softwareentwicklung

Aufgabe MSE:
48 Punkte

14. Automaten

a) Gegeben sei der nachfolgende Automat. Der Automat befindet sich aktuell im Zustand *s4*.
Leiten Sie eine Übersicht der Übergänge ab, indem sie die untenstehende Tabelle vervollständigen (Zustand und Ausgabe)!

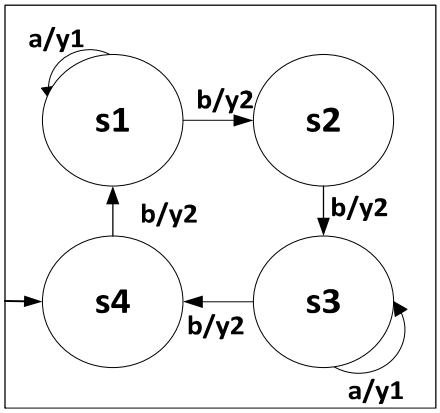


Bild MSE-14.1: *Automat*

T	s1	s2	s3	s4
a				
b				

Welche Eingabe müssen Sie tätigen damit Sie die Ausgabesequenz

y2 y1 y1 y2 erhalten?

_____

In welchem Zustand befindet sich der Automat nach einer weiteren Ausgabe von

y2 y1 y1 y2 der Eingabe?

_____

Handelt es sich um einen deterministischen oder nicht-deterministischen Automaten?

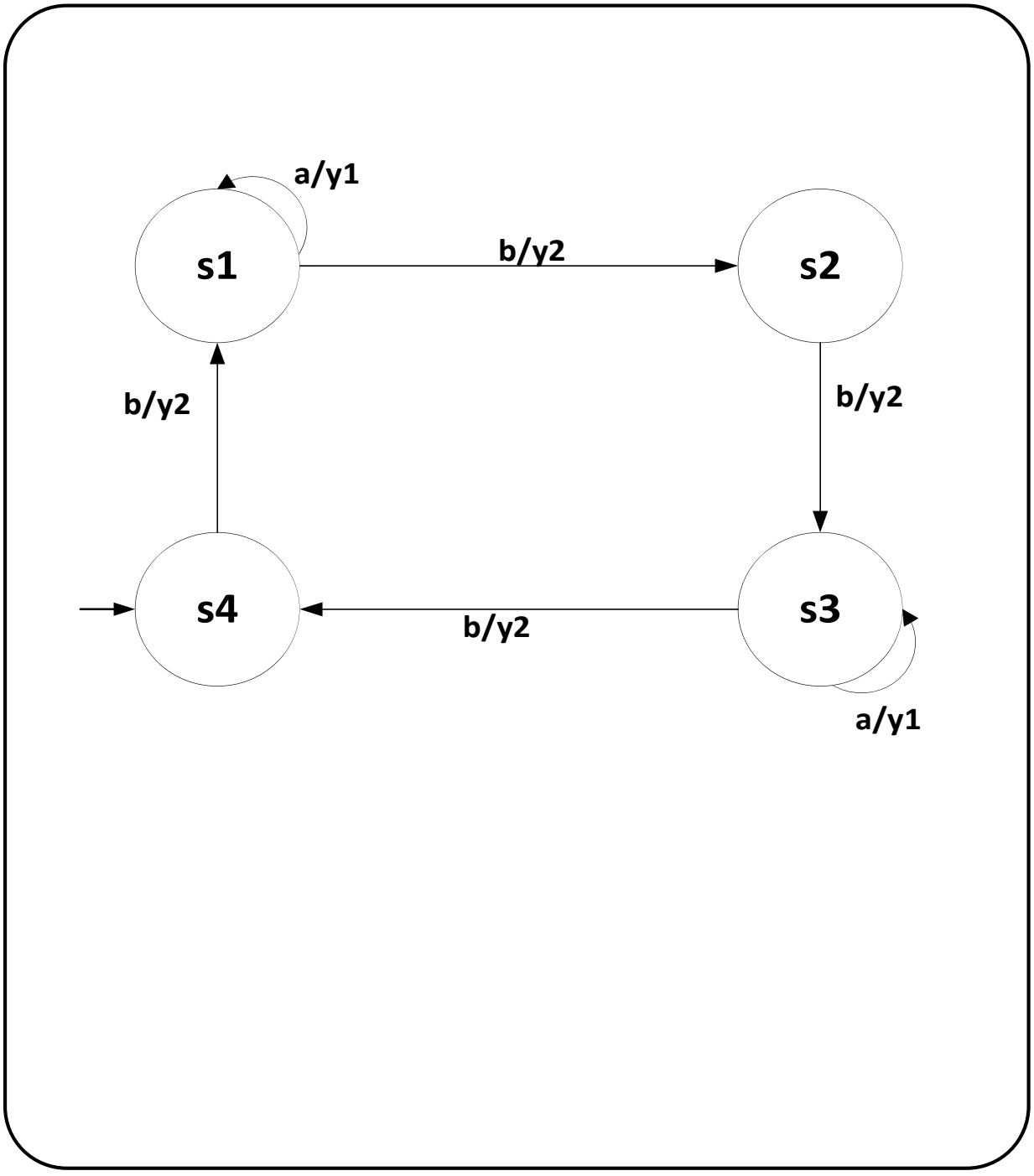
Begründen Sie Ihre Antwort! (Begründung ausschlaggebend)

_____

b) Der Automat aus a) repräsentiert die Betriebsarten einer Maschine. Im nachfolgenden soll der Automat erweitert werden:

- Der Anfangszustand *s4* bleibt bestehen.
- Durch Eingabe von a und b und c gleichzeitig, also *abc*, kann die Maschine aus den Zuständen *s4* und *s2* in den Zustand *sPause* übergehen. In beiden Fällen wird *y00* ausgegeben.
- Das Verlassen des Zustands *sPause* ist nur durch den Übergang zum Anfangszustand *s4* möglich. Hierfür ist eine Eingabe *c* notwendig. Ausgegeben wird dabei *y2.*

Vervollständigen Sie den Automaten gemäß der obigen Beschreibung!



15. Zustandsdiagramm

Die Funktionsweise eines Sparschweins soll mithilfe eines Zustandsdiagramms (gemäß der Vorlesung) modelliert werden, welches ausschließlich in der Lage ist 1€-Münzen ist anzunehmen.

Das Sparschwein kann *leer*, *teilweise gefüllt* oder *voll* sein. Zu *Beginn* ist dieses *leer*. Durch die Aktion *füllen* kann es mit Münzen *teilweise gefüllt* werden. Das maximale Fassungsvermögen des Sparschweins umfasst 50€ - das Sparschwein ist dann also *voll*. Der Inhalt des Sparschweins kann durch die Aktion *leeren* jederzeit reduziert werden. Das *füllen* bzw. *leeren* wird durch die Sensoren (boolesche Variablen) *bEinwurf* bzw. *bAuswurf* erfasst. Die Anzahl der Münzen im Sparschwein wird durch die Integer-Variable *zähler* festgehalten. Der Wert dieser Variablen wird bei *füllen* automatisch inkrementiert bzw. bei *leeren* automatisch dekrementiert. Das Inkrementieren / Dekrementieren muss nicht modelliert werden. Der Zustand des Sparschweins wird durch eine leuchtende Nase mithilfe einer LED, angezeigt: Ist das Sparschwein *voll*, dann wird durch die Aktion *Leuchten* die Nase *rot* erstrahlen. In allen anderen Zuständen leuchtet diese in *grüner* Farbe. Die Farbe wird dabei der Aktion *Leuchten* als Parameter übergeben.

Hinweis: Verwenden Sie zur Überprüfung der Anzahl der Münzen die Variable *zähler*. Damit die LED dauerhaft leuchtet, muss die jeweilige Aktion dauerhaft aufgerufen werden

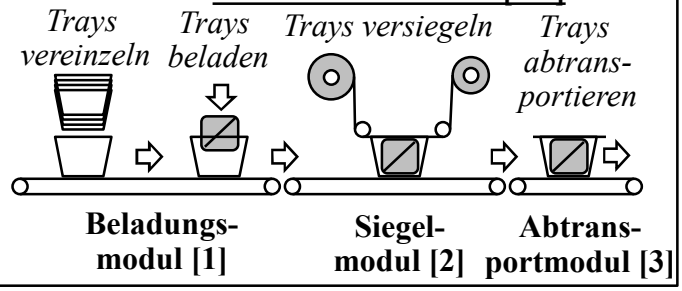
16. SA/RT: Flussdiagramm

Für die folgenden Teilaufgaben ist eine Maschine zur Verpackung und Versiegelung von Sandwiches in Plastikschalen (Trays) gegeben, die aus drei Modulen besteht (Bild G-16.1).

Im *Beladungsmodul* (Modul 1) werden leere Trays aus einem Traystapel vereinzelt und danach mit Sandwiches befüllt. Im *Siegelmodul* (Modul 2) werden die Trays mit einer Oberfolie versiegelt. Im *Abtransportmodul* (Modul 3) werden die fertig versiegelten Trays an den nächsten Prozessschritt weitergegeben, der hier nicht weiter betrachtet wird.

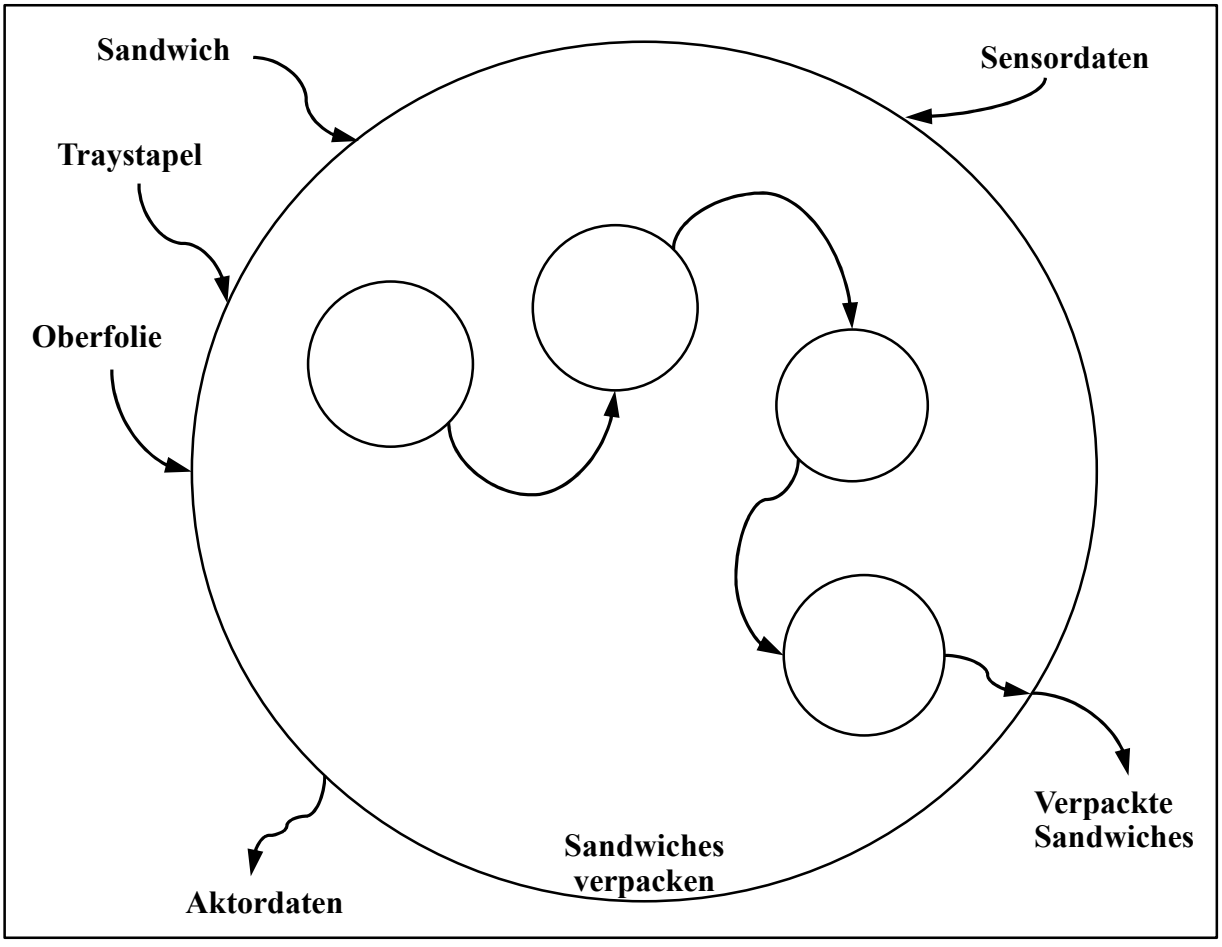
Das System soll in den folgenden Aufgaben mittels Strukturierter Analyse/ Real-Time (SA/RT) modelliert werden. Beachten Sie dabei folgende Informationen:

Bild MSE-16.1: *Prozesse* und **Module [Nr.]**



Flüsse:
Material: Traystapel, leere Tray, volle Tray, verpacktes Sandwich, Sandwich, Oberfolie
Sensordaten: je nach Modul: SD*Modulnummer*, also z.B. SD2 bei Siegelmodul.
Aktordaten: Wie Sensordaten nur AD*Modulnummer*

Modellieren Sie den Prozess *Sandwiches verpacken* in einem *Datenflussdiagramm*. Identifizieren Sie hierzu alle Subprozesse und Datenflüsse und tragen Sie diese mit Bezeichnung ein. Beachten Sie, dass Sensor- und Aktordaten eines Moduls mit SD*Modulnummer* und AD*Modulnummer* (z.B. SD1 oder AD3) zusammengefasst werden.



17. Antwortzeitspezifikation: Timing-Diagramm

Zur Verfeinerung des Prozesses *Tray abtransportieren* soll eine Antwortzeitspezifikation in Form eines Timing-Diagramms erstellt werden, um sicherzustellen, dass der Prozess korrekt durchgeführt wird. Rechts sehen sie eine detailliertere Darstellung des Abtransportmoduls (Bild MSE-17.1), welches diesen Prozess umsetzt.

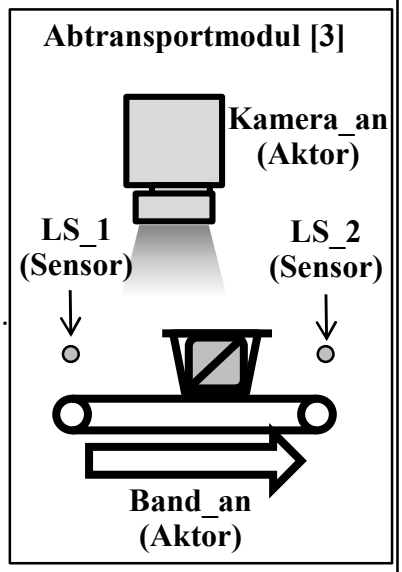
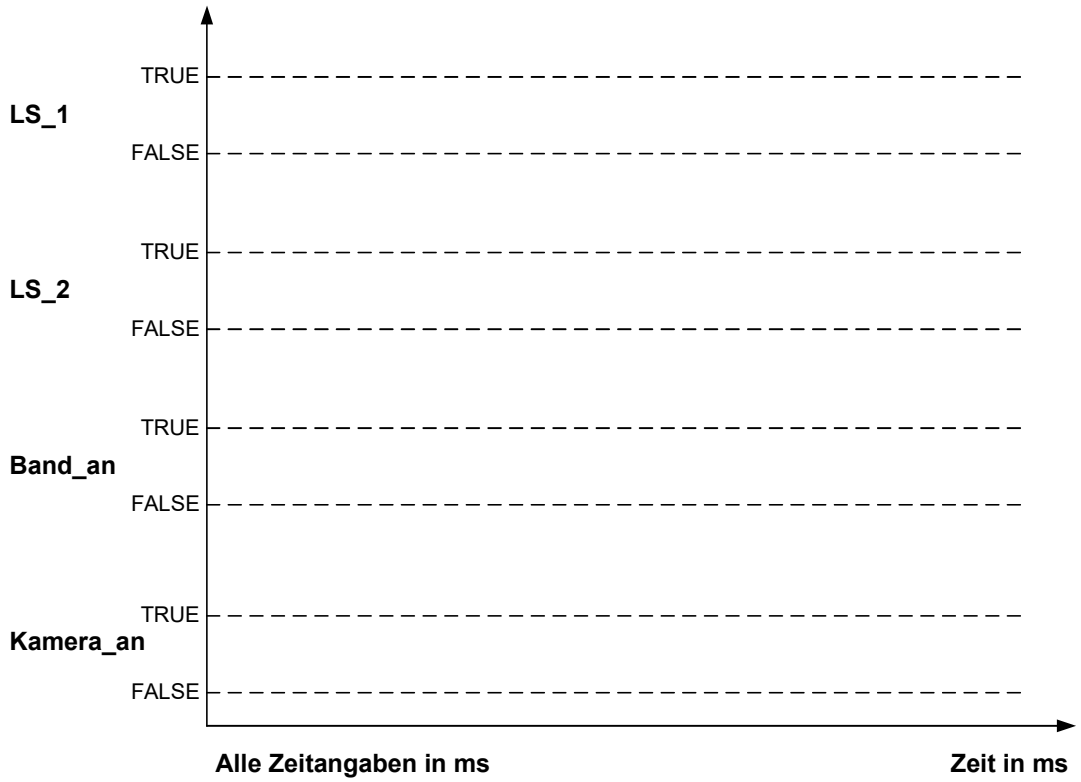


Bild MSE-17.1: *Modul*

Die Werte der Sensoren und Aktoren können jeweils *TRUE* (z. B. Tray erkannt) oder *FALSE* (z. B. Tray nicht erkannt) sein.

Ergänzen Sie das Timing-Diagramm gemäß folgender Angaben (Werteverläufe und Zeitangaben):

- Sobald eine Tray auf dem Band erkannt wird, liefert die Lichtschranke *LS_1* den Wert TRUE, woraufhin die Motorvariable *Band_an* innerhalb von 100 +/− 50 ms auf TRUE geht.
- Nachdem das Band an ist, springt zum einen die Lichtschranke nach 200 +/- 100 ms wieder auf FALSE (Tray ist durch Lichtschranke hindurchgefahren), zum anderen wird die Kamera über die Aktorvariable *Kamera_an* innerhalb 150 +/- 100 ms aktiviert.
- 3 +/- 0.1 s nach Anschalten des Bandes erkennt LS_2 den Tray (TRUE)
- Es dauert 150 +/- 50 ms bis die Tray durch LS_2 hindurchgefahren ist, woraufhin sowohl das Band als auch die Kamera nach 200 +/-100 ms wieder ausgeschaltet werden.



Aufgabe C-Programmierung und MATLAB/Simulink

Aufgabe C:
96 Punkte

18. C: Datentypen

Definieren Sie die Datentypen der folgenden Variablen so, dass so wenig Speicher wie möglich benötigt wird. Die Variablen sollen zur Beschreibung eines Aufzugs benutzt werden.

Variable `masse`, aktuelle Last auf eine Nachkommastelle genau

Variable `maxPers`, Maximale Personenzahl, maximal 1000

Variable `pruefNext`, Nächstes Prüfdatum, codiert in der Form MMYYY (MM = Monat, YYYY = Jahr)

Variable `pruefNr`, Eindeutige Prüfnummer mit einer Länge von 14 Ziffern

Kreuzen Sie an, welchen Datentyp Sie jeweils verwenden würden (nur Einfachnennung möglich).

18.1 `masse`

( ) `char` ( ) `int`
( ) `long` ( ) `float`
( ) `double`

18.2 `maxPers`

( ) `char` ( ) `int`
( ) `long` ( ) `float`
( ) `double`

18.3 `pruefNext`

( ) `char` ( ) `int`
( ) `long` ( ) `float`
( ) `double`

18.4 `pruefNr`

( ) `char` ( ) `int`
( ) `long` ( ) `float`
( ) `double`

19. C: Dateneingabe

Sie wollen die Variable **`masse`** nun durch den Benutzer eingeben lassen. Ergänzen Sie die Lücken mit den Nummer 1 bis 2 um die Eingabe korrekt einzulesen.

```
scanf( __(1)__, __(2)__ );
```

Lücke 1:
( ) `"%i"` ( ) `"%f"`
( ) `"%d"` ( ) `"&d"`
( ) `"%e"`

Lücke 2:
( ) `*masse` ( ) `masse[]`
( ) `masse` ( ) `= masse`
( ) `&masse`

20. C: Boolesche Algebra

Bestimmen Sie das Ergebnis der nachfolgend angegebenen Ausdrücke im Dezimalsystem. Gegeben seien dafür folgende Variablen:

```
int x = 15;
int y = 3;
float z = 3.14f;
char c = 0x63;
int* pj;
pj = &y;
```

Nach jedem der nachfolgenden Ausdrücke werden die Variablen auf die oben genannten Werte zurückgesetzt.

20.1 `((x/y)>>2) && (x%y)`

20.2 `(y++*3) || *pj`

20.3 `(c/15) ^ y`

20.4 `*pj*(((int)z)<z && !(x%y)))<<2`

21. C: Kontrollstrukturen

Sie sollen für die Verkehrsbetriebe einer bekannten deutschen Stadt Daten zur Verspätung von einzelnen U-Bahnlinien berechnen. Hierzu liegen Daten zur Verspätung (Variable *verspaetung*) der einzelnen Linien U1 bis U3 (*LIN*) an vier Stationen (*STAT*) auf der Strecke vor (Angabe in Minuten). Ihre Aufgabe ist es nun, die Verspätung der einzelnen Linien in der Variable *summeLinVerspaetung* aufzusummieren und die durchschnittliche Verspätung der Linie in die Variable *avgLinVerspaetung* zu speichern. Weiterhin wünschen die Verkehrsbetriebe Informationen über die gesamte Verspätung aller Linien an allen Station (*summeVerspaetung*). Davon ausgehend soll die durchschnittliche Verspätung im Netz (*avgVerspaetung*) abgeleitet werden. Weiterhin soll bei der Verspätung einer Linie an einer Station größer als 2 Minuten eine Meldung ausgegeben werden.

Hinweis: Verwenden Sie für Ihr C-Programm die angegebene Vorlage (Sourcecode C-21.1). Kreuzen Sie für die jeweiligen im Sourcecode C-21.1 angegebenen Lücken mit den Nummern 1 bis 10 auf der folgenden Seite an, welchen C-Code Sie einfügen. Jeweils nur Einfachnennung möglich.

```
#include ____①____
#define LIN  3
#define STAT 4
int main()
{                                       // Stationen          Linien
    float verspaetung____②_____    = {{ 0  , 1.1, 2.5, 1.3},  // U1
                                      { 5.0, 4.8, 1.2, 0  },  // U2
                                      { 0  , 0.3, 0  , 0.1}}; // U3
    float summeLinVerspaetung[LIN] =  { 0  , 0  , 0};  // Initalisieren
    float avgLinVerspaetung[LIN]   =  { 0  , 0  , 0};
    float summeVerspaetung = 0;
    float avgVerspaetung = 0;
    int i = 0; int j;
    for(____③____)        // Iterieren über Linien LIN
    {
        j = 0;
        while(_____④_____)      // Iterieren über Stationen STAT
        {
            _____⑤_____ (_____⑥_____)    // Wenn Verspaetung zu groß
            {
                _____⑦_____("Linie U%d hatte an Station %d zu viel Verspaetung!\n",
                        _____⑧_____); // Gebe Meldung aus
            }
            summeLinVerspaetung[i] += verspaetung[i][j];     // Aufsummieren
            summeVerspaetung += verspaetung[i][j];           // Aufsummieren
            _____⑨_____;
        }
        avgLinVerspaetung[i] = summeLinVerspaetung[i] / STAT;
    }
    avgVerspaetung = summeVerspaetung / LIN / STAT;
    ____⑩____;
}
```

Sourcecode C-21.1

21.1 Lücke **Nr.1 im Sourcecode C-21.1**

( ) `io.h` ( ) `<stdio>` ( ) `<stdio.h>`

( ) `<stdlib.h>` ( ) `stdlib` ( ) `define <io.h>`

21.2 Lücke **Nr.2 im Sourcecode C-21.2**

( ) `(STAT)(LIN)` ( ) `[STAT,LIN]` ( ) `{STAT}{LIN}`

( ) `(LIN)(STAT)` ( ) `[LIN][STAT]` ( ) `{LIN*STAT}`

21.3 Lücke **Nr.3 im Sourcecode C-21.2**

( ) `i=0; i+1;i<LIN` ( ) `i=0;i<LIN;i++` ( ) `i=1;i<LIN;i++`

( ) `i=0;i<=LIN;i+1` ( ) `i=0;++i;i<LIN` ( ) `i<<LIN;i+=1`

21.4 Lücke **Nr.4 im Sourcecode C-21.2**

( ) `j<<STAT` ( ) `i<<STAT` ( ) `LIN++`

( ) `i<STAT` ( ) `j<STAT` ( ) `j<LIN`

21.5 Lücke **Nr.5 im Sourcecode C-21.2**

( ) `for` ( ) `while` ( ) `do`

( ) `switch` ( ) `if` ( ) `case`

21.6 Lücke **Nr.6 im Sourcecode C-21.2**

( ) `verspaetung[i][j]>2` ( ) `verspaetung(j)(i)>2`

( ) `verspaetung[j;i]==2` ( ) `verspaetung[i,j]>2`

( ) `verspaetung[j,i] >2` ( ) `verspaetung(i,j)>2`

21.7 Lücke **Nr.7 im Sourcecode C-21.2**

( ) `fputs` ( ) `println` ( ) `scanf`

( ) `fgetc` ( ) `echo` ( ) `printf`

21.8 Lücke **Nr.8 im Sourcecode C-21.2**

( ) `i,j` ( ) `j,i` ( ) `%i,%j`

( ) `i+1,j+1` ( ) `&i,&j` ( ) `i++,j++`

21.9 Lücke **Nr.9 im Sourcecode C-21.2**

( ) `j = j<<1` ( ) `j++` ( ) `j+1`

( ) `j--` ( ) `*j++` ( ) `j = j | 0x01`

21.10 Lücke **Nr.10 im Sourcecode C-21.2**

( ) `return` ( ) `break` ( ) `continue`

( ) `return 0` ( ) `break 0` ( ) `continue 0`

22. C: Zyklische Programmierung eines Mischbehälters

Für eine prototypische Anlage (siehe Bild C-22.1) der Firma MischMasch AG muss der Steuerungscode in C geschrieben werden. Die Anlage umfasst einen Misch- und Reaktionsbehälter **B1**. In diesen können über die Ventile **V1** und **V2** die jeweiligen Edukte (Ausgangsstoffe) *Edukt 1* und *Edukt 2* zu dosiert werden. Während der Dosierung soll der Rührer am Behälter **B1** aktiviert werden. Nach Abschluss der Reaktion muss der Behälter entleert und der Rührer abgeschaltet werden. Die einzelnen Zustände 0 bis 4 sollen zyklisch abgearbeitet werden.

Eine Liste aller relevanten Sensoren, Aktoren und Variablen finden Sie in Bild C-22.2. Beachten Sie, dass sich Ventil V3 anders als V1 und V2 verhält.

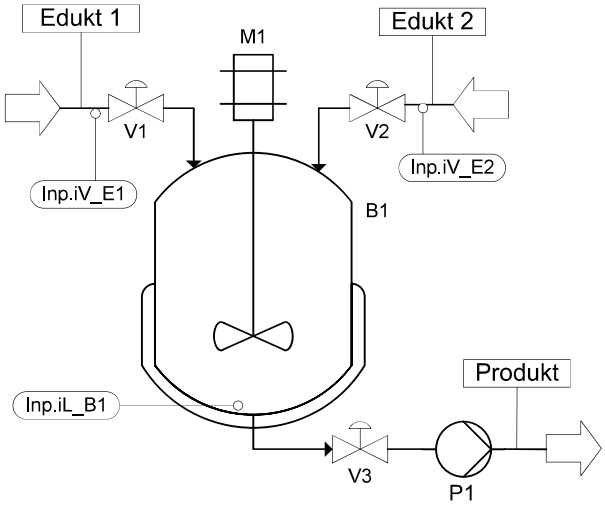


Bild C-22.1: *Skizze des Misch- und Reaktionsbehälters*

Typ	Name	Beschreibung
Aktor	`Out.oM1`	Aktiviert(1), Deaktiviert(0) den Rührer **M1** an Behälter **B1**
	`Out.oP1`	Schaltet Pumpe **P1** ein(1) oder aus (0)
	`Out.oV1`	Öffnet(1) oder schließt(0) das Ventil **V1** (Edukt 1)
	`Out.oV2`	Öffnet(1) oder schließt(0) das Ventil **V2** (Edukt 2)
	`Out.oV3`	Öffnet(0) oder schließt(1) das Ventil **V3** (Produkt)
Sensor	`Inp.iL_B1`	Füllstand des Behälters **B1**: leer(0), nicht leer(1)
	`Inp.iV_E1`	Zudosiertes Volumen an Edukt 1 in Litern
	`Inp.iV_E2`	Zudosiertes Volumen an Edukt 2 in Litern
Variablen	`active`	Steuert die Ausführung des Programms (Start/Stop)
	`Rez.VE1S`	Dosiervolumen an Edukt 1, welches zudosiert werden soll in Litern
	`Rez.VE2S`	Dosiervolumen an Edukt 2, welches zudosiert werden soll in Litern
	`Rez.tRuehr`	Gesamte Rührzeit ab Einschalten laut Rezept in ganzen Sekunden
	`status`	Speichert den aktuellen Zustand
	`timer`	Stoppuhr mit den vergangenen ganzen Sekunden ab Einschalten

Bild C-22.2: *Aktoren, Sensoren und Variablen*

Im folgenden sind die einzelnen Zustände der Anlage (siehe Bild C-22.3) gegeben:

Zustand 0: Nach Beginn des Programms wird der Rührer **M1** eingeschalten. Zusätzlich wird die Dosierung von Edukt 1 gestartet. Hierfür muss das entsprechende Ventil geöffnet werden.

Zustand 1: Nach der erfolgten Dosierung der im Rezept festgelegten Menge an Edukt 1 wird für die im Rezept vorgegebene Zeit weitergerührt.

Zustand 2: Ist die vorgegebene Rührzeit erreicht, soll bei weiter laufendem Rührer Edukt 2 über das Ventil **V2** zudosiert werden.

Zustand 3: Ist die vergebene Menge Edukt 2 fertig dosiert wird der Tank leer gepumpt. Der Schalter **`Inp.iL_B1`** gibt an, ob sich noch Flüssigkeit im Tank befindet. Ist der Tank leer wird der Vorgang gestoppt. Während des gesamten Vorgangs läuft der Rührer weiter.

Zustand 4: Dieser Zustand dient zum korrekten Abschalten der Anlage. Die noch aktivierten Komponenten werden zurückgesetzt (ausgeschalten), und das Programm beendet. Weiterhin soll über die Konsole eine Ausgabe in der Form

```
Rezept nach 36 Sekunden abgeschlossen.
```

erscheinen.

22.1 Zustandsdiagramm

Vervollständigen Sie die Übergangsbedingungen zwischen den beschriebenen Zuständen im Zustandsdiagramm in Bild C-22.3.

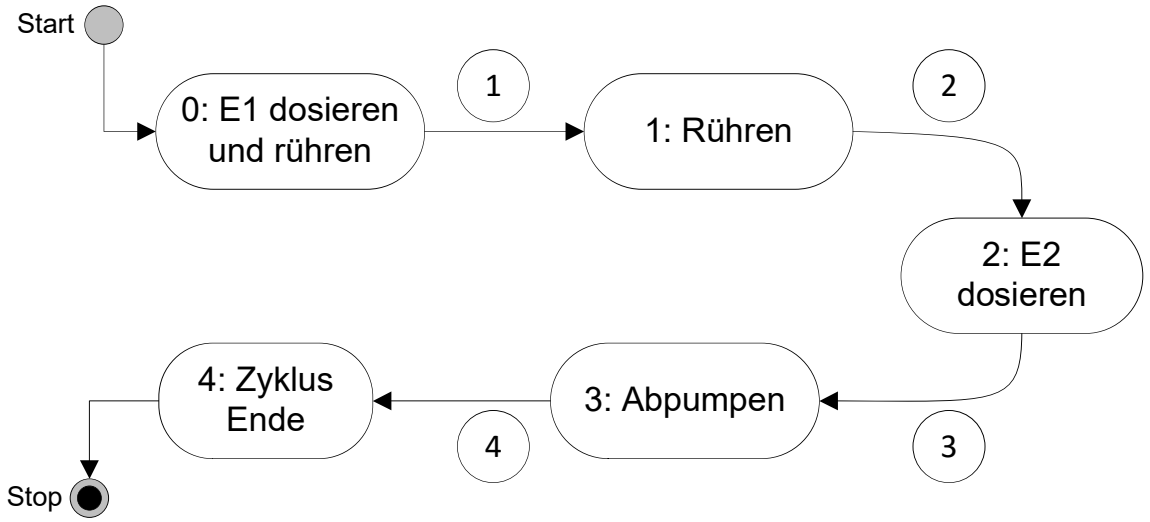


Bild C-22.3: *Zustandsdiagramm der Dosieranlage*

Lücke 1: _____

Lücke 2: _____

Lücke 3: _____

Lücke 4: _____

22.2 Zyklische Programmierung

Vervollständigen Sie nun den vorgebenen Programmcode (siehe Bilder C-22.4 und C-22.5). Beachten Sie, dass die Funktion `rezeptLesen` *by reference* aufgerufen wird. Weiterhin soll das Programm die Bearbeitungszeit eines Rezepts in Sekunden als Rückgabewert zurückgeben.

```
#include "IO_MixingPlant.h"
#include <stdio.h>
_______ main()
{
    Sensor Inp;                 // Sensorvariablen
    Actuator Out;               // Aktorvariablen
    Rezept Rez;                 // Rezeptdaten
    char status = 0;            // Aktueller Zustand
    char active = 1;            // Programm laueft solange true
    int timer;                  // Zeit in Sekunden seit Start
    initClock(&timer);          // Setze Startzeitpunkt
    while (active)              // Zyklische Ausfuehrung
    {
        rezeptLesen(_________);// Einlesen des Rezepts
        updateClock(&timer);
        switch (___________)
        {
        _____________:         // Zustand 0
            _________________ // Aktoren setzen
            _________________
            if (_____________________________)
            {
                _________________    // Aktor setzen
                _________________    // 0->1
            }
            _________________
      _____________:           // Zustand Ruehren
             ____________________
                _____________// 1->2
            _________________
       _____________:          // Zustand E2 dosieren
            _________________   // Aktor setzen
            ______________________________ {
                _____________  // Aktor setzen
                _____________ }// 2->3
            _________________
```

Bild C-22.4: *Programmcode der Dosieranlage (Teil 1 von 2)*

```
        _______________:            // Zustand Abpumpen
              ___________________   // Aktoren setzen
              ___________________
              ___________________
              {
                   ___________________    // Aktoren setzen
                   ___________________
                   ___________________    // 3->4
              }
              ___________________

       _______________:             // Ende Zyklus
              ___________________   // Aktor setzen
              ___________________   // Zyklische Ausfuehrung beenden
              ____________("Rezept nach ___ Sekunden abgeschlossen.",
                   ________);//Ausgabe der Zeit für Rezeptabarbeitung
              ___________________
         }
    }
    return_________________         // Programm beenden
}
```

Bild C-22.5: *Programmcode der Dosieranlage (Teil 2 von 2)*

23. C: Structs

Zur Speicherung von Prozessdaten sollen geeignete Datenfelder angelegt werden. Hierbei müssen zwei speichersparende Struktur-Datentypen definiert werden.

Die Struktur `Messwert` soll einen ganzzahligen UNIX-Zeitstempel `lTimestamp` und den aktuellen Durchfluss `fDurchfluss` als Fließkommazahl aufnehmen. `lTimestamp` kann hierbei sehr groß werden und liegt beispielsweise für den 01.01.2017 bei 1.483.228.800.

Ein weiterer Strukturdatentyp mit dem Namen `Messstelle` soll eine Idendtifikationsnummer `ID` speichern. Des Weiteren soll ein Array mit dem Namen `wert` vom Typ `Messwert` gespeichert werden. Dieses Array soll eine Länge gleich der Konstante `VALUES` haben.

Die besprochenen Datentypen sind in Bild C-23.1 zusammengefasst. Vervollständigen Sie den in Bild C-23.2 gezeigten Inhalt der Headerdatei *messwerte.h* um die Typendefinitionen:

```
Messwert:
   lTimestamp:  0-2.000.000.000
   fDurchfluss: Fließkommazahl
```

```
Messstelle:
   ID:    0-50.0000
   wert:  Array vom Typ Messwert
          mit Länge VALUES
```

Bild C-23.1: *Benötigte Datentypen, deren Variablen und typische Wertebereiche*

```
#define VALUES 2

____________________

{
     ______ lTimestamp;
     float fDurchfluss;
} ____________________

____________________
{
    unsigned int ID;
    ____________________________________;
} ____________________
```

Bild C-23.2: *Inhalt der Headerdatei messwerte.h*

24. Dateioperationen

Die Messwerte aus der Dosieranlage werden von einem separaten Archivprogramm für die spätere Analyse in der Datei *Import.csv* gespeichert. Bei dieser Datei handelt es sich um eine Datei, welche die einzelnen Werte getrennt von einem Semikolon (Strichpunkt) speichert (vgl. hierzu Bild C-24.1). Jede Zeile beinhaltet zwei Messungen an einer Messstelle in der Form

`ID;Zeitstempel1;Messwert1;Zeitstempel2;Messwert2\n`

```
1;1489741200;3.820000;1489741230;2.600000
23;1489741203;25.299999;1489741300;17.815100
```

Bild C-24.1: *Inhalt der Archivdatei Import.csv*

Sie sollen nun ein Konvertierungsprogramm schreiben, welches die Daten aus der *Import.csv* ausliest und in einer Array des Typs Messstelle speichert. Anschließend soll der Inhalt dieses Arrays im Binärformat in die Datei *Export.bin* exportiert werden. Vervollständigen Sie den Quellcode in Bild C-7.2, um die Daten korrekt in die in Bild C-24.2 definierten Datentypen einzulesen. Zunächst müssen Sie eine lokale Variable der Länge `SENSORS` definieren, welche die Daten der einzelnen Messstellen aufnimmt. Anschließend öffnen Sie die Datei *Import.csv* und lesen die Daten korrekt in die zuvor definierte Variable ein. Im nächsten Schritt, schreiben Sie den Inhalt der Variablen im Binärmodus in die Datei *Export.bin*.

```
#include <stdio.h>
#include "messwerte.h"
#define SENSORS 2
int main()
{
    Messstelle Daten[_________________];   // Deklaration des Arrays

    __________________ = ________("Import.csv","rt"); // File-Handle
    for (_____________________________)       // Zeilenweises Einlesen
    {
        fscanf(__________,"_________________________",&Daten[_].ID,
      &Daten[_].wert[_].lTimestamp,&Daten[_].wert[_].fDurchfluss,
      &Daten[_].wert[_].lTimestamp,&Daten[_].wert[_].fDurchfluss);
      // Einlesen einer Zeile und Zuweisung an Array
    }
    ____________________________          // Schließen der Datei
    __________________ = ________("Export.bin",____); // File-Handle
    _______(Daten,sizeof(Messstelle),SENSORS,_______); //Schreiben
    ____________________                  // Schließen der Datei
    return 0;
}
```

Bild C-24.2: *Einlesen der Daten aus der Datei Import.csv*

25.1 Matlab/Simulink

Gegeben ist folgender Aufbau eines Schaltnetzes. Zur Überprüfung des Verhaltens soll diese in einem Matlab/Simulink Projekt simulativ untersucht werden. Das dafür vorgesehene Testsignal wurde in einem „Signal Builder“ Block erstellt. Als Simulationszeit gilt: Fixed-Step Size mit 1 sec

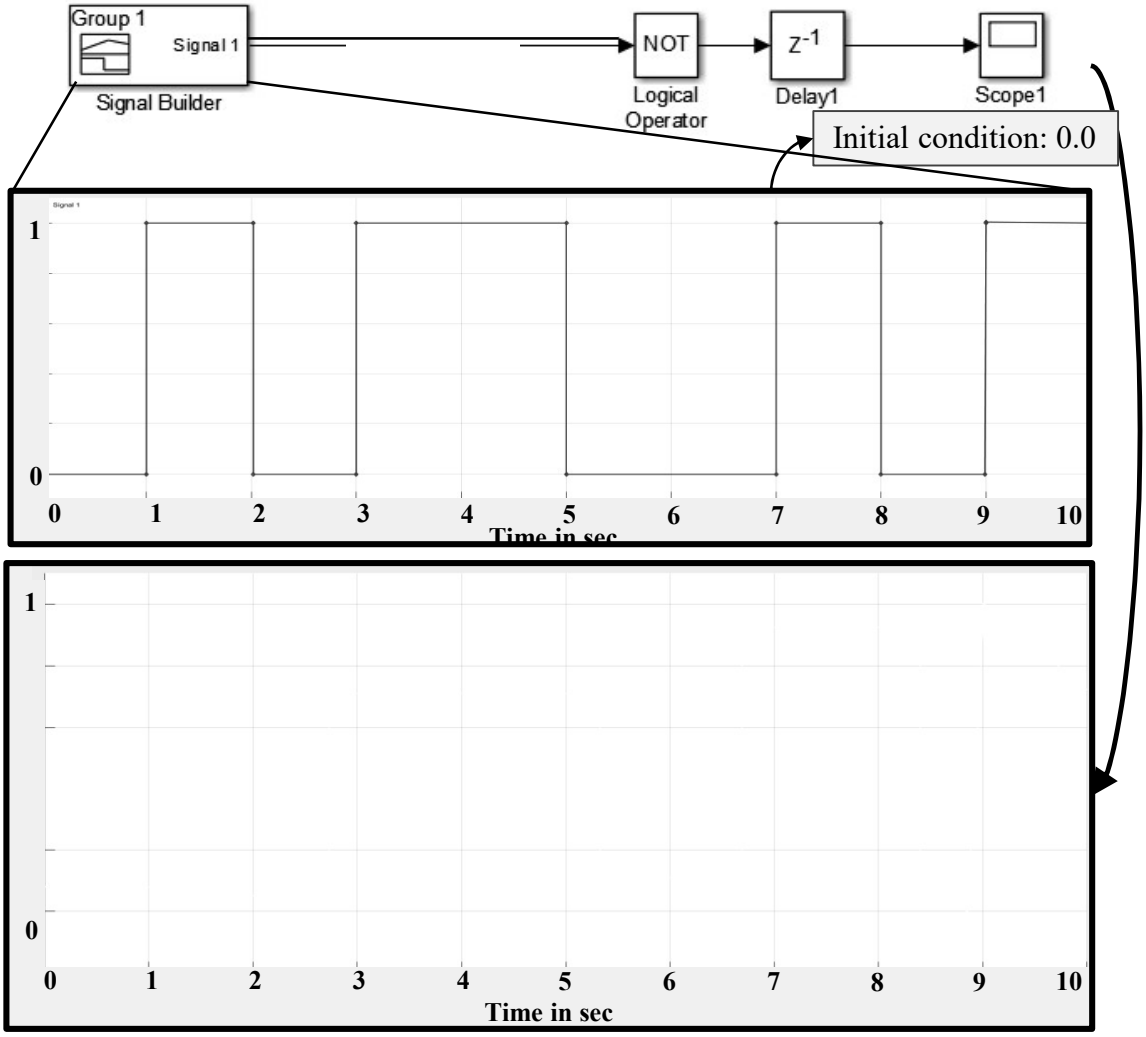


Bild C-25.1: *Aufbau Schaltnetz*

Zeichnen Sie das zu erwartende Signal von Scope 1 ein.

25.2 Matlab/Simulink

Vervollständigen Sie das untenstehende Modell so, dass die folgende Funktion abgebildet wird: **sin(a) - 12**

Bild C-25.2: *Aufbau Modell*

25.3 Matlab/Simulink

Eine Anlage zum Trennen von Werkstücken (WS) (siehe Bild) soll von einem Zustandsautomaten in Stateflow gesteuert werden. Die zu trennenden WS werden über ein Zulieferband zu einer Kranstation transportiert und anschließend mit Hilfe von zwei Abfuhrbändern getrennt. Für die Detektion eines WS besitzt jedes Band am Eingang eine Lichtschranke (*iLSZ*, *iLSA1*, *iLSA2*). Ein Scanner am Zulieferband erkennt die ID des WS (1 oder 2), sodass der Kran die WS getrennt auf Abfuhrband 1 (ID 1) und Abfuhrband 2 (ID 2) ablegen kann. Für die Detektion der Position besitzt der Kran den Endlagenpositionssensor *iKranPos* mit den Werten 0 (Zulieferband), 1 (Abfuhrband1) und 2 (Abfuhrband2). Das Drehen des Krans in Uhrzeigerrichtung erfolgt über die Variable *oKranL*. Zum Aufnehmen und Ablegen eines WS besitzt der Kran die Aktoren *oKranWSAuf* und *oKranWSAb*. Die Sensoren *iLSA1* und *iLSA2* liegen direkt unterhalb des Ablageplatzes des Krans. Die Dauer für das Aufnehmen eines WS beträgt 2 Sekunden.

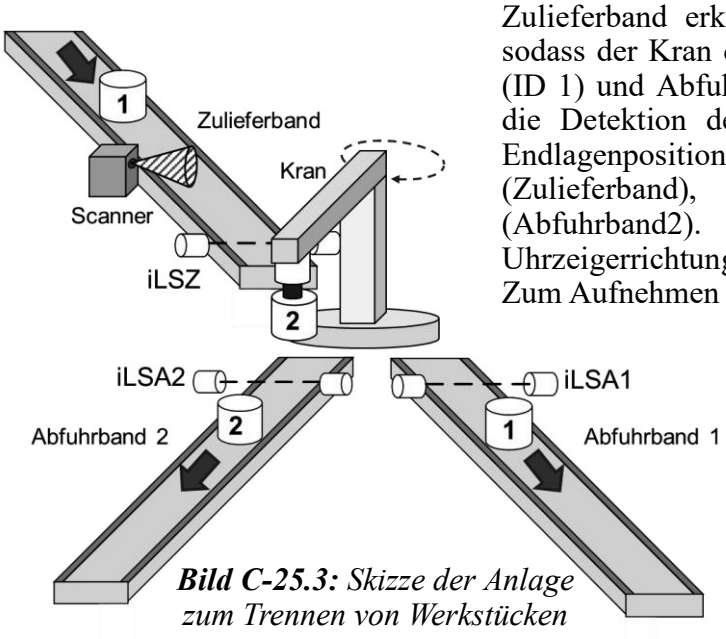


Bild C-25.3: *Skizze der Anlage zum Trennen von Werkstücken*

Sensoren und Aktoren

Aktoren	oMotorZ	Bool	Schaltet den Motor des Zulieferbandes an (true) und aus (false)
	oMotorA1	Bool	Schaltet den Motor des Abfuhrbandes 1 an (true) und aus (false)
	oMotorA2	Bool	Schaltet den Motor des Abfuhrbandes 2 an (true) und aus (false)
	oKranL	Bool	Kran nach links drehen (true) oder stoppen (false)
	oKranWSAuf	Bool	Kran nimmt WS auf (true)
	oKranWSAb	Bool	Kran legt WS ab (true)
Sensoren	iLSZ	Bool	Bauteil erkannt am Eingang des Zulieferbandes (true); Wenn kein Bauteil erkannt (false)
	iLSA1	Bool	Bauteil erkannt am Eingang des Abfuhrbandes1 (true); Wenn kein Bauteil erkannt (false)
	iLSA2	Bool	Bauteil erkannt am Eingang des Abfuhrbandes2 (true); Wenn kein Bauteil erkannt (false)
	iScanner	Int	ID des Werkstücks (1) oder (2)
	iKranPos	Int	Endlagen Position des Krans – Zulieferband (0), Abfuhrband1 (1) und Abfuhrband2 (2)

Vervollständigen Sie den unten dargestellten Zustandsautomaten in Stateflow, sodass alle Anforderungen aus der Aufgabenstellung erfüllt sind. Verwenden Sie die unten angegebenen Transitionsbedingungen und ordnen Sie diese durch die Verwendung der Buchstaben (A…H) dem Zustandsdiagramm zu.

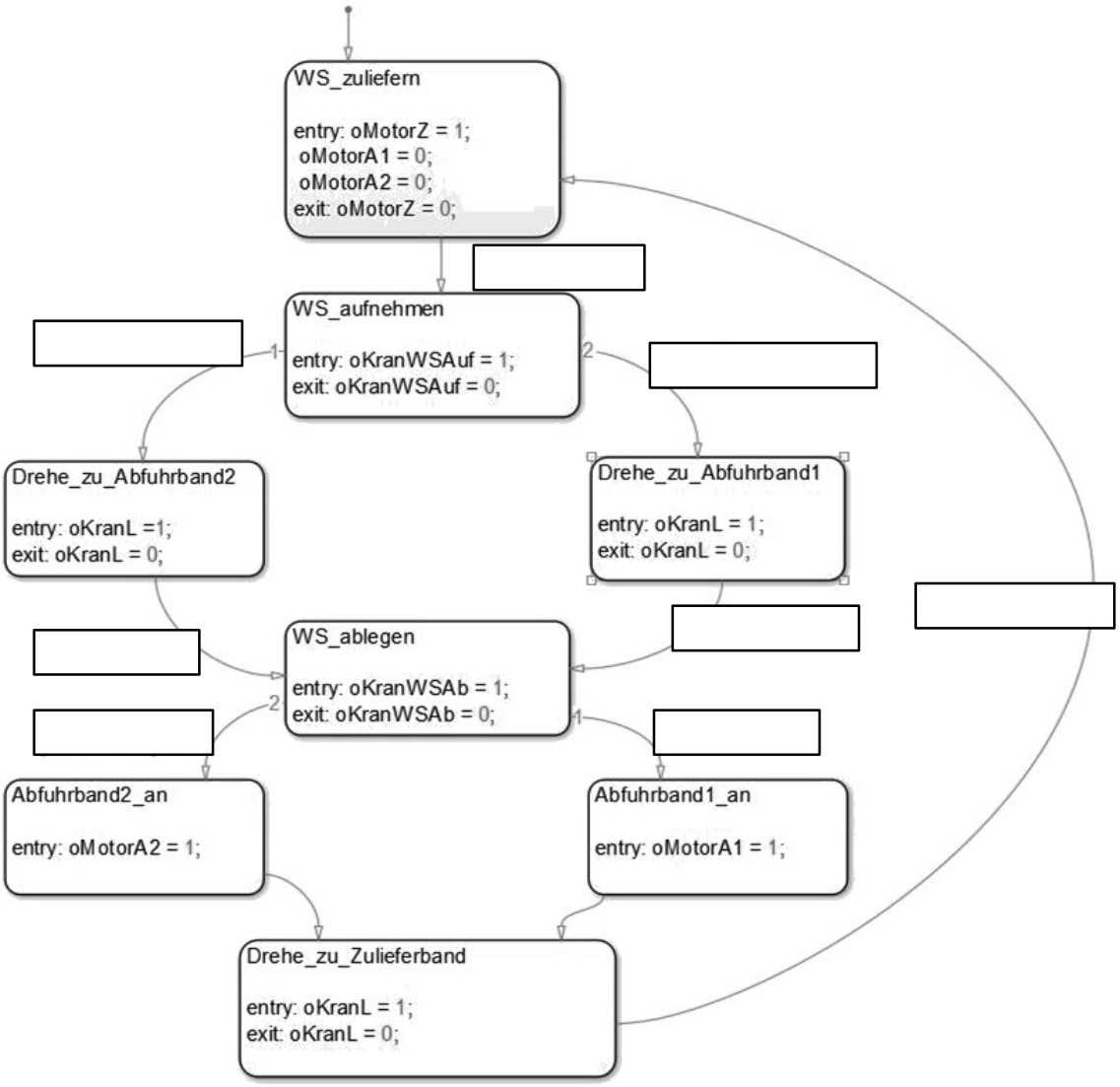


Bild C-25.4: *Stateflow*

A **[iLSA1==1]**

B **[iLSA2 == 1]**

C **[iLSZ==1]**

D **[after (2, sec) && iScanner==1]**

E **[after (2, sec) && iScanner==2]**

F **[iKranPos==0]**

G **[iKranPos==2]**

H **[iKranPos==1]**