Prof. Dr.-Ing. Jürgen Teich
Lehrstuhl für Informatik 12
(Hardware-Software-Co-Design)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Klausur
Grundlagen der Technischen Informatik

4. April 2019

Vorname	
Nachname	
Matrikelnummer	

Aufgabe	1	2	3	4	5	Σ
Max. Punkte	16	16	16	16	16	80
Erreichte Punkte						
Note						

Organisatorische Hinweise

Bitte sorgfältig lesen und die Kenntnisnahme durch Unterschrift bestätigen

---

a) Bitte legen Sie Ihren Studentenausweis bereit.

b) Als Hilfsmittel sind nur Schreibmaterialien und ein beidseitig handbeschriebenes DIN A4-Blatt zugelassen.

c) Verwenden Sie weder Rot- noch Bleistifte.

d) Sie können bei der Aufsicht zusätzliche Bearbeitungsblätter anfordern.

e) Unleserliches wird nicht bewertet.

f) Bei mehreren präsentierten Lösungen wird die Aufgabe nicht gewertet. Streichen Sie daher bei Angabe mehrerer Lösungsansätze die nicht zu bewertenden Lösungen durch.

g) Die Bearbeitungszeit beträgt 120 Minuten.

Erklärung

a) Im Falle einer während der Prüfung auftretenden Prüfungsunfähigkeit zeige ich dies sofort der Aufsicht an und befolge deren Anweisungen. Ich weiß, dass ich die volle Beweislast trage. Ich lasse mir das Formular des Prüfungsamts, das für diese Fälle vorgesehen ist, aushändigen und verfahre nach den dort niedergelegten Richtlinien.

b) Ich weiß, dass im Falle des Täuschungsversuchs oder der Benutzung unerlaubter Hilfsmittel („Unterschleif“) der Prüfungsausschuss die Entscheidung treffen kann, die betroffene Prüfungsleistung als mit „nicht ausreichend“ bewertet gelten zu lassen.

c) Ich habe die obigen Hinweise zur Kenntnis genommen.

Erlangen, den .....

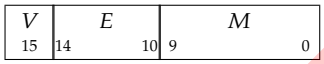
Unterschrift

Aufgabe 1 (Zahlensysteme) (16 Punkte)

a) Wie lautet der Wertebereich einer n-stelligen vorzeichenlosen Quintalzahl (Zahl im Fünferzahlensystem)? (2 Punkte)

b) Wie lautet der Wertebereich einer n-stelligen Binärzahl im Zweierkomplement? (2 Punkte)

c) In dieser Aufgabe soll mit 16-Bit Gleitkommazahlen gearbeitet werden. Diese werden analog zum IEEE-Format gebildet. Das Format der Gleitkommazahl sieht dabei wie folgend aus:
Vorzeichen (*1 Bit*), *Exponent* (*5 Bit*), *Mantisse* (*10 Bit*)



Führen Sie nun die Multiplikation der beiden in diesem Format dargestellten Gleitkommazahlen 0 11000 1110000000 und 1 00110 0100100000 aus, und geben Sie das Ergebnis im gleichen Format an. (7 Punkte)

d) In dieser Aufgabe wird ein 4-Bit Mikrocontroller betrachtet, welcher nur arithmetische Operationen für vorzeichenlose 4-Bit Binärzahlen ausführen kann. Somit stehen Ihnen die arithmetischen Operationen op $\in \{+, -, =, \neq, >, \geq, <, \leq\}$, die logischen Verknüpfungen op $\in \{\wedge, \vee\}$, sowie die Negation $\neg x$ zur Verfügung. Die Operanden x und y sowie das Ergebnis $r = x \,\mathrm{op}\, y$ sind 4-Bit Binärzahlen. Ergebnisse für vier Beispiele der $>$-Operation sind im Folgenden gegeben.

Operation					Ergebnis
$S(1100_2) =$	12_{10}	$>$	$S(1010_2) =$	10_{10}	1
$S(1100_2) =$	12_{10}	$>$	$S(0110_2) =$	6_{10}	1
$S(0111_2) =$	7_{10}	$>$	$S(0110_2) =$	6_{10}	1
$S(0111_2) =$	7_{10}	$>$	$S(1010_2) =$	10_{10}	0

Da die $>$-Operation leider nur für vorzeichenlose 4-Bit Binärzahlen ausgelegt ist, können Operanden im 4-Bit Zweierkomplement nicht korrekt verglichen werden. Im Folgenden sind die obigen vier Beispieloperationen nochmals aufgelistet, wenn deren Operanden nun 4-Bit Zweierkomplementzahlen darstellen.

Operation					Ergebnis
$S_2(1100_2) =$	-4_{10}	$>$	$S_2(1010_2) =$	-6_{10}	1
$S_2(1100_2) =$	-4_{10}	$>$	$S_2(0110_2) =$	6_{10}	1 korrekt wäre 0
$S_2(0111_2) =$	7_{10}	$>$	$S_2(0110_2) =$	6_{10}	1
$S_2(0111_2) =$	7_{10}	$>$	$S_2(1010_2) =$	-6_{10}	0 korrekt wäre 1

Geben Sie einen Test an, welcher zu **wahr** auswertet (1), wenn der Wert $S_2(x)$ der 4-Bit Zweierkomplementzahl x größer als der Wert $S_2(y)$ der 4-Bit Zweierkomplementzahl y ist. Dem Test stehen ausschließlich die 11 Operationen des 4-Bit Mikrocontrollers, Konstanten darstellbar als vorzeichenlose 4-Bit Binärzahlen, $x = (x_3,x_2,x_1,x_0)$ und $y = (y_3,y_2,y_1,y_0)$, sowie beliebige Zwischenergebnisse zur Verfügung. (5 Punkte)

Hinweis: Um zu überprüfen, ob eine 4-Bit Zweierkomplementzahl x negativ ist, können Sie den Test $S(x) \geq 8$ verwenden.

Aufgabe 2 (Minimierung von Schaltfunktionen) (16 Punkte)

Für eine Verschlüsselung soll eine Schaltung zur Umwandlung von UTF-8-kodierten Buchstaben entworfen werden. Gegeben sei dafür ein Auszug der UTF-8-Kodierungstabelle:

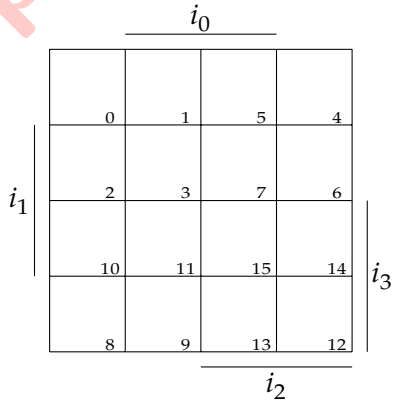
Code (0x)	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o

Der Buchstabe *a* ist demnach kodiert als Hexadezimalwert 0x61, darstellbar mit 8 Bits. Die Kodierung eines eingehenden Buchstabens $i_7 i_6 i_5 i_4 i_3 i_2 i_1 i_0$ soll nach einer festgelegten Vorschrift in ein Ausgangssymbol $o_7 o_6 o_5 o_4 o_3 o_2 o_1 o_0$ wie folgt umkodiert werden. Für die Verschlüsselung genügt es, die 4 niederwertigsten Bits der UTF-8-kodierten Symbole zu betrachten:

i_3	i_2	i_1	i_0	Eingangsbuchstabe	Ausgangssymbol	o_3	o_2	o_1	o_0
0	0	0	1	a	^				
0	0	1	0	b	[				
0	0	1	1	c	]				
0	1	0	0	d	Z				
0	1	0	1	e	\				
0	1	1	0	f	Y				
0	1	1	1	g	X				
1	0	0	0	h	U				
1	0	0	1	i	W				
1	0	1	0	j	T				
1	0	1	1	k	V				
1	1	0	0	l	S				
1	1	0	1	m	R				
1	1	1	0	n	Q				
1	1	1	1	o	P				

a) Vervollständigen Sie die Funktionstabelle der Verschlüsselungsfunktion. (1 Punkt)

b) Übertragen Sie o_2 in das gegebene Symmetriediagramm, und geben Sie eine KMF für o_2 an. (2 Punkte)



c) Ermitteln Sie alle Primimplikanten für o_2 unter Verwendung des Nelson-Verfahrens. (4 Punkte)

d) Stellen Sie eine Überdeckungstabelle der Primimplikanten von o_2 auf und ermitteln Sie alle kostengünstigsten DMFs für o_2 unter Verwendung des Petrick-Verfahrens. (3 Punkte)

		j								
k	PI								p_i	c_i
0									A	
1									B	
2									C	
3									D	

e) Sei $o_1(i_3, i_2, i_1, i_0) = (\overline{i_2} + \overline{i_1}) \cdot (\overline{i_3} + i_2 + i_0) \cdot (i_3 + (\overline{i_1} + \overline{i_0}) \cdot (\overline{i_2} + \overline{i_0}))$ eine Schaltfunktion für o_1. Stellen Sie sowohl das Pull-Up-Netzwerk, als auch das Pull-Down-Netzwerk einer CMOS-Schaltung für $o_1(i_3, i_2, i_1, i_0)$ auf. (2 Punkte)

f) Zeichnen Sie eine CMOS-Schaltung für $o_1(i_3, i_2, i_1, i_0)$ unter Verwendung Ihrer aufgestellten Pull-Up- und Pull-Down-Netzwerke in das gegebene Diagramm ein. Die Eingangssignale i_3, i_2, i_1 und i_0 liegen dabei nur in nicht-negierter Form vor. (4 Punkte)
Hinweis: Zur besseren Übersicht verwenden Sie Teilnetzbeschriftungen, z. B. $\overline{i_3}$.

VDD

GND

Aufgabe 3 (Automaten) (16 Punkte)

a) Im Folgenden soll ein Automat entworfen werden, der die Motorsteuerung einer Aufzugstür realisiert. Für den Antrieb der Aufzugstür wird ein Elektromotor eingesetzt, der abhängig von seiner Drehrichtung das Öffnen und Schließen der Aufzugstür ermöglicht. Während im Regelbetrieb die Aufzugstür nicht manuell geöffnet bzw. geschlossen werden kann, soll der Motor beim Eintreten eines Notfalls das manuelle Bewegen der Tür zulassen. Der Automat verfügt über vier verschiedene Eingaben, die jedoch nicht gleichzeitig anliegen können, sowie vier verschiedene Ausgaben.

Eingaben:

- Die Aufzugstür soll geöffnet werden (Eingabe *I_Open*).
- Die Aufzugstür soll geschlossen werden (Eingabe *I_Close*).
- Die Position der Aufzugstür ist in Endstellung (Eingabe *I_Hold*).
- Ein Notfall ist eingetreten (Eingabe *I_Emergency*).

Ausgaben:

- Der Motor öffnet die Tür (Ausgabe *M_Open*).
- Der Motor schließt die Tür (Ausgabe *M_Close*).
- Der Motor hält die aktuelle Position der Aufzugstür (Ausgabe *M_Hold*).
- Der Motor ermöglicht das manuelle Bewegen der Aufzugstür (Ausgabe *M_Manual*).

Die Ein- und Ausgaben sind wie folgt durch binäre Variablen codiert:

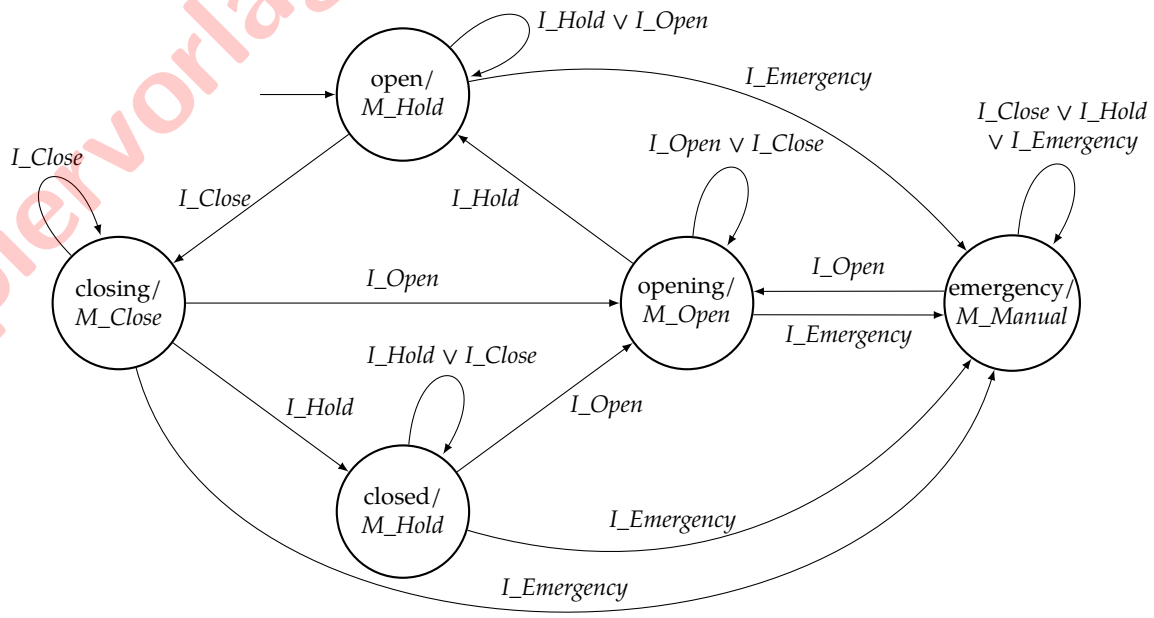
Eingabe	i_1	i_0
I_Open	0	0
I_Close	0	1
I_Hold	1	0
I_Emergency	1	1

Eingaben des Automaten

Ausgabe	o_1	o_0
M_Open	0	0
M_Close	0	1
M_Hold	1	0
M_Manual	1	1

Ausgaben des Automaten

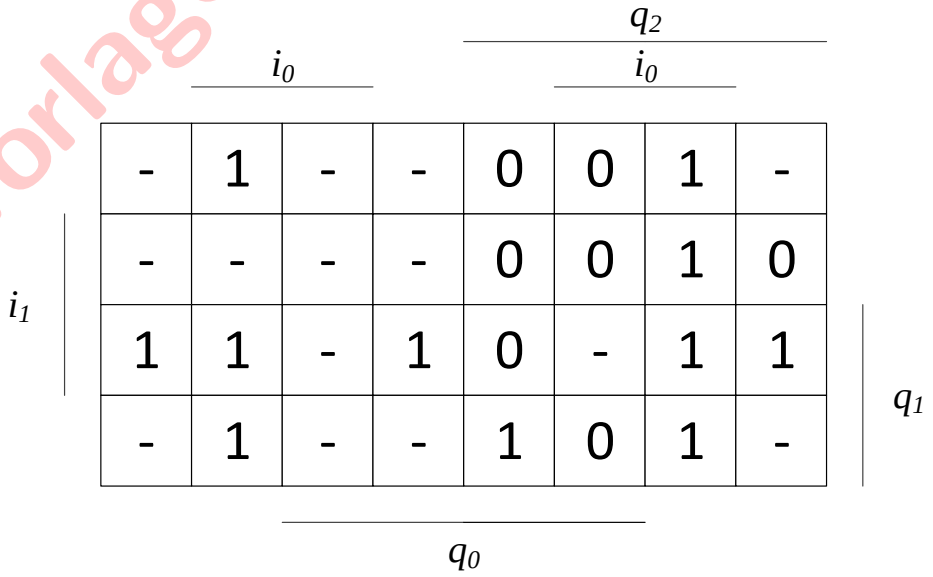
Das Verhalten des Automaten ist in folgendem Automatengraphen spezifiziert:



1. Vervollständigen Sie die nachfolgend gegebene Automatentafel unter Verwendung von taktflankengesteuerten D-, JK- bzw. T-Flipflops. (6 Punkte)

Zustandsname	Aktueller Zustand			Eingabe		Nachfolgezustand			Ansteuerfunktion				Ausgabe	
	q_2	q_1	q_0	i_1	i_0	q_2'	q_1'	q_0'	D_2	J_1	K_1	T_0	o_1	o_0
open	0	0	0	0	0									
open	0	0	0	0	1									
open	0	0	0	1	0									
open	0	0	0	1	1									
closing	0	0	1	0	0									
closing	0	0	1	0	1									
closing	0	0	1	1	0									
closing	0	0	1	1	1									
closed	0	1	0	0	0									
closed	0	1	0	0	1									
closed	0	1	0	1	0									
closed	0	1	0	1	1									
opening	0	1	1	0	0									
opening	0	1	1	0	1									
opening	0	1	1	1	0									
opening	0	1	1	1	1									
emergency	1	0	0	0	0									
emergency	1	0	0	0	1									
emergency	1	0	0	1	0									
emergency	1	0	0	1	1									

2. Nehmen Sie im Folgenden an, dass die Ansteuerfunktion K_1 des JK-Flipflops im unten gegebenen Symmetriediagramm spezifiziert sei. Entwickeln Sie eine KMF der Ansteuerfunktion K_1, und geben Sie den resultierenden schaltalgebraischen Ausdruck an.
Hinweis: Die hier angegebene Ansteuerfunktion K_1 entspricht nicht der Lösung von Teilaufgabe 1.
(3 Punkte)



3. Zeichnen Sie ein Schaltwerk für den in Teilaufgabe 1 realisierten Automaten. Verwenden Sie dazu die in Teilaufgabe 2 bestimmte KMF der Ansteuerfunktion K_1, und gehen Sie davon aus, dass Sie die benötigten Ansteuerfunktionen D_2, J_1 und T_0 als Eingangssignale zur Verfügung haben. Auf die Realisierung der Ausgabe des Automaten kann verzichtet werden. (3 Punkte)

b) Geben Sie den Typen des in Teilaufgabe a) spezifizierten Automaten an. (1 Punkt)

c) Geben Sie den allgemeinen Aufbau eines Medwedew-Automaten als Blockschaltbild an. (2 Punkte)

d) Zeichnen Sie ein Schaltnetz, das für ein Eingangssignal C sowohl positive als auch negative Flankenübergänge erkennt und einen Impuls mit der Dauer von 1 ns am Ausgang C' generiert. Die Verzögerungszeit jedes Logikgatters in dieser Aufgabe betrage $\tau = 1\,\text{ns}$. (1 Punkt)

Aufgabe 4 (Codierung und Rechnerarithmetik) (16 Punkte)

a) DNA besteht aus den vier Nukleobasen *Adenin* (A), *Guanin* (G), *Cytosin* (C) und *Thymin* (T). Bei der DNA-Sequenzierung werden Nukleobasen in sogenannte Standardtriplets zusammengefasst, beispielsweise GCC. In folgender Tabelle sind die in einer konkreten DNA-Sequenz vorhandenen Triplets und deren Auftrittshäufigkeit angegeben.

Triplets	GCC	CTA	ATG	ACC	TAG	$\sum$
Auftrittshäufigkeit	8	1	2	4	1	16

1. Geben Sie die Gleichung für die Entropie an und berechnen Sie die Entropie der in der obigen Tabelle dargestellten Quelle. (2 Punkte)

2. Ordnen Sie den Informationsgehalt der folgenden Aussagen in aufsteigender Reihenfolge. (2 Punkte)

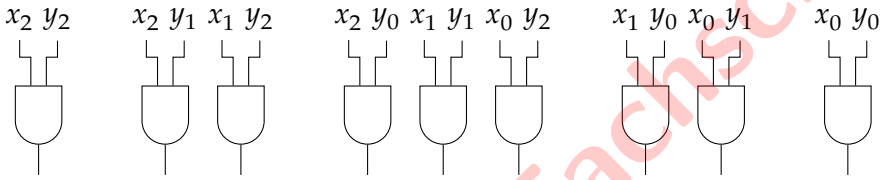
   A:= „Das Triplet ist TAG“
   B:= „Das Triplet enthält mindestens ein C“
   C:= „Das Triplet beginnt mit A“

3. Erstellen Sie einen Huffman-Codierungsbaum für die obige Quelle. (3 Punkte)

b) Im Folgenden wird eine sogenannte Wallace-Tree-Multiplikation zweier 3-Bit Binärzahlen $x_2x_1x_0 \cdot y_2y_1y_0 = s_5s_4s_3s_2s_1s_0$ beschrieben. Dabei werden die Partialprodukte x_iy_j mit $0 \leq i,j < 3$ nicht wie üblich zeilenweise, sondern (wie im folgenden Beispiel gezeigt) spaltenweise aufaddiert.

$$
\begin{array}{lcccccc}
x_2x_1x_0 \cdot y_2y_1y_0 = & & & & & & \\
 & & & \boxed{x_1y_2} & \boxed{x_0y_2} & \boxed{x_0y_1} & x_0y_0 \\
+ & & & x_2y_1 & x_1y_1 & x_1y_0 & \\
+ & & x_2y_2 & & x_2y_0 & & \\
\hline
= & s_5 & s_4 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

1. Entwerfen Sie das Schaltnetz eines Wallace-Tree-Multiplizierers, welcher zwei 3-Bit Zahlen $x_2x_1x_0 \cdot y_2y_1y_0 = s_5s_4s_3s_2s_1s_0$ miteinander multipliziert. Dazu stehen Ihnen ausschließlich Volladdierer und Halbaddierer zur Verfügung. Vervollständigen Sie die folgende Schaltung mit den entsprechenden Bausteinen. (4 Punkte)



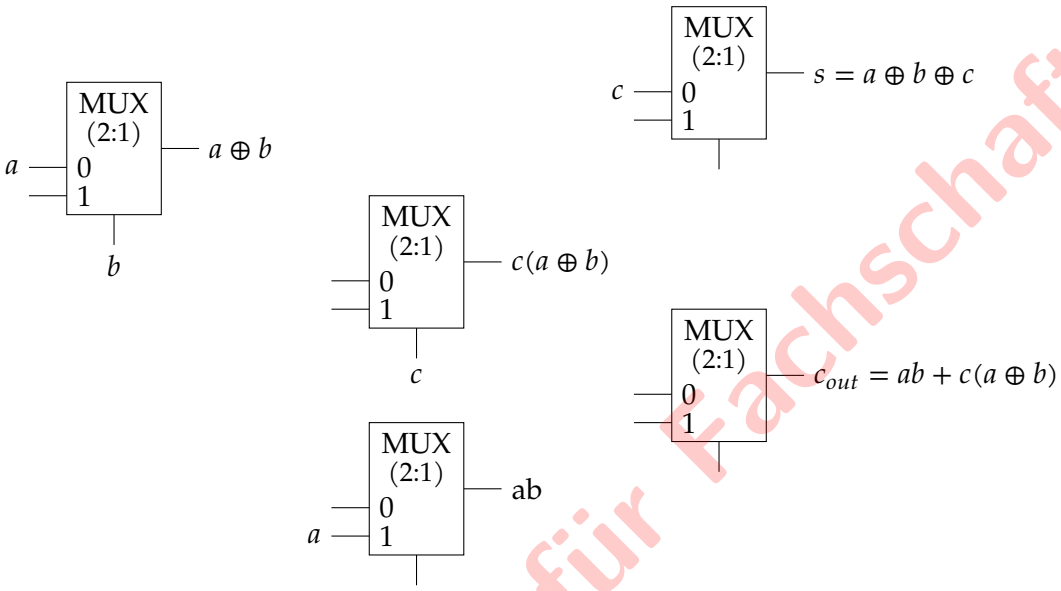
s_5 s_4 s_3 s_2 s_1 s_0

2. Wieviele UND-Gatter werden zur Erzeugung der Partialprodukte (x_iy_j mit $0 \leq i,j < 3$) bei einem sequentiellen Multiplizierer benötigt?

(1 Punkt)

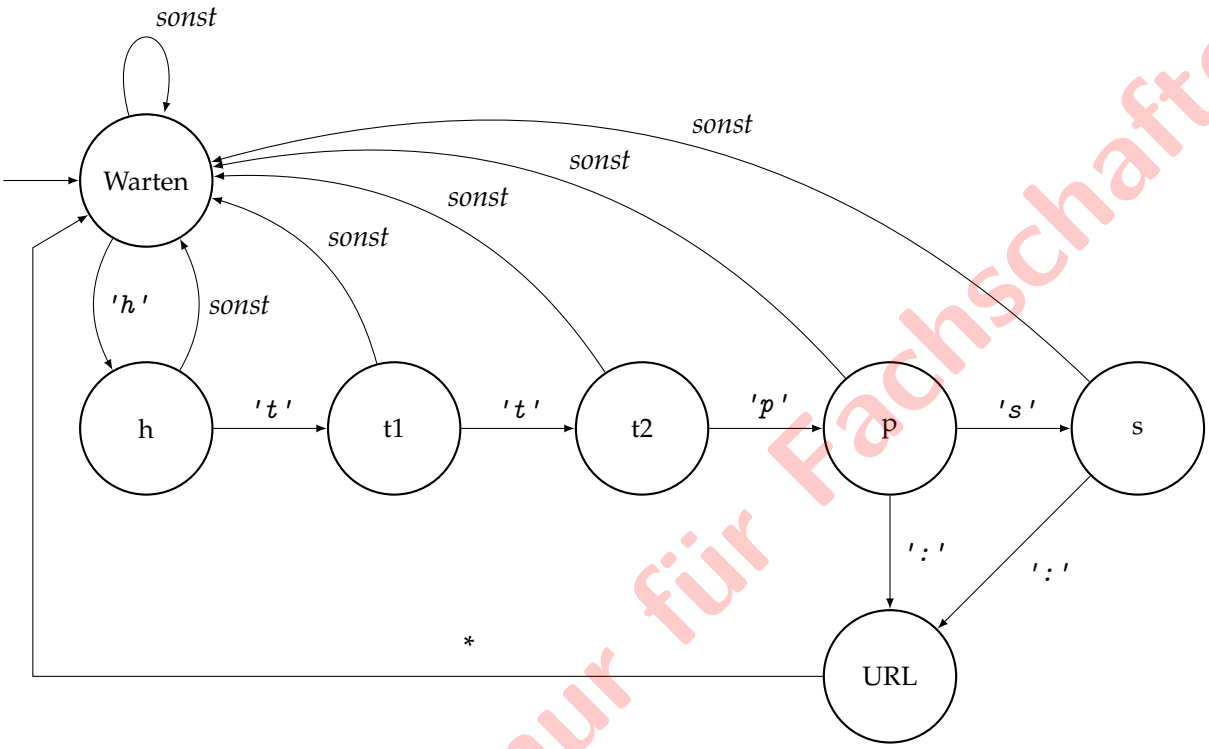
c) Realisieren Sie einen Volladdierer unter ausschließlicher Verwendung von fünf 2:1-Multiplexern. Der Volladdierer verfügt über die Eingangssignale a, b und c und erzeugt die Ausgangssignale s (Summe) und c_{out} (Übertrag). In folgender Schaltung sind die zu verwendenden Multiplexer und deren Ausgangssignale bereits gegeben. Vervollständigen Sie diese Schaltung durch geeigneter Verknüpfung der Ein- und Ausgänge der Multiplexer sowie Verwendung der Signale $a, \overline{a}, b, \overline{b}, c, \overline{c}$ sowie $0, 1$.

(4 Punkte)



Aufgabe 5 (VHDL) (16 Punkte)

Implementieren Sie einen sogenannten Sniffer (eine Komponente, die Kommunikation überwacht), der URLs in einem stetigen Zeichenstrom erkennt. Es sollen URLs unterstützt werden, die entweder mit `http:` oder mit `https:` beginnen. Diese Funktionalität wird durch den im folgenden Automatengraphen beschriebenen Automaten implementiert:



Implementieren Sie im Folgenden den spezifizierten Automaten als *synchrone* Schaltung, die sich ebenfalls *synchron* zurücksetzen lässt, in VHDL. Im Zustand URL soll ein Ausgangssignal `is_url` auf 1 gesetzt werden, sonst stets auf 0 bleiben. Das eingehende Zeichen werde mit `input` bezeichnet. Weiterhin stehe folgender Datentyp zur Verfügung:

```
type sniffer_state is (warten, h, t1, t2, p, s, url);
```

a) Vervollständigen Sie folgende `entity`-Deklaration. Alle Ein- und Ausgänge sollen vom Typ `std_logic` oder `std_logic_vector` sein. Das Eingabezeichen werde als `character` (8-Bit ASCII-Zeichen) codiert. (2 Punkte)

```
entity sniffer is
  port(




  );
end sniffer;
```

b) Vervollständigen Sie folgende Funktion, die für eine gegebene Kombination von Zustand und Eingabe den nächsten Zustand zurückliefert. (4 Punkte)

```
function get_next_state(state: sniffer_state, input: character)
    return sniffer_state is
begin




end function;
```

c) Vervollständigen Sie folgende Funktion, die zurückliefert, ob eine URL erkannt wurde. (1 Punkt)

```
function is_done(state: sniffer_state) return std_logic is
begin

end function;
```

d) Vervollständigen Sie den Rumpf der folgenden `architecture`-Beschreibung unter Verwendung der beiden obigen Funktionen. (3 Punkte)

```
architecture behavioral of sniffer is
  signal state : sniffer_state;
  signal input_char : character;
  signal is_url_reg : std_logic;
begin
  -- vorgegeben: Umwandlung von std_logic_vector nach character
  input_char <= convert_to_ascii(input);
  is_url <= is_url_reg;

end architecture;
```

e) Vervollständigen Sie folgende Funktion, die zurückgibt, wie oft eine gegebene Ziffer `digit` in einem beliebig langen Bitvektor `arg` vorkommt. (Hinweis: `natural` ist ein Typ, der die natürlichen Zahlen inklusive der 0 darstellt.) (3 Punkte)

```
function count(digit: std_logic, arg: std_logic_vector)
    return natural is




end function;
```

f) Erläutern Sie die Unterschiede zwischen Signalen und Variablen. (2 Punkte)

g) Wie funktioniert eine Testbench? (1 Punkt)