Aufgabe 1.1:Technologien und Grundlagen *(12,5 Punkte)*

a) Gegeben sei folgende Wahrheitstabelle:

a	**b**	**c**	***f(a,b,c)***
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Geben Sie $f(a, b, c)$ in disjunktiver kanonischer Normalform (DKN) an.

b) Vereinfachen Sie $f(a, b, c)$ algebraisch soweit wie möglich unter Anwendung der Axiome der boolschen Algebra.

c) Formen Sie $f(a, b, c)$ so um, dass sich die Schaltfunktion ausschließlich aus NAND-Ausdrücken über jeweils zwei Termen zusammensetzt.

d) Gegeben ist die in Abbildung 1 dargestellte unvollständige CMOS-Schaltung. Es ist lediglich die Pullup-Schaltung abgebildet. Bestimmen Sie in einem ersten Schritt welche logische Funktion diese Pullup-Schaltung angibt.

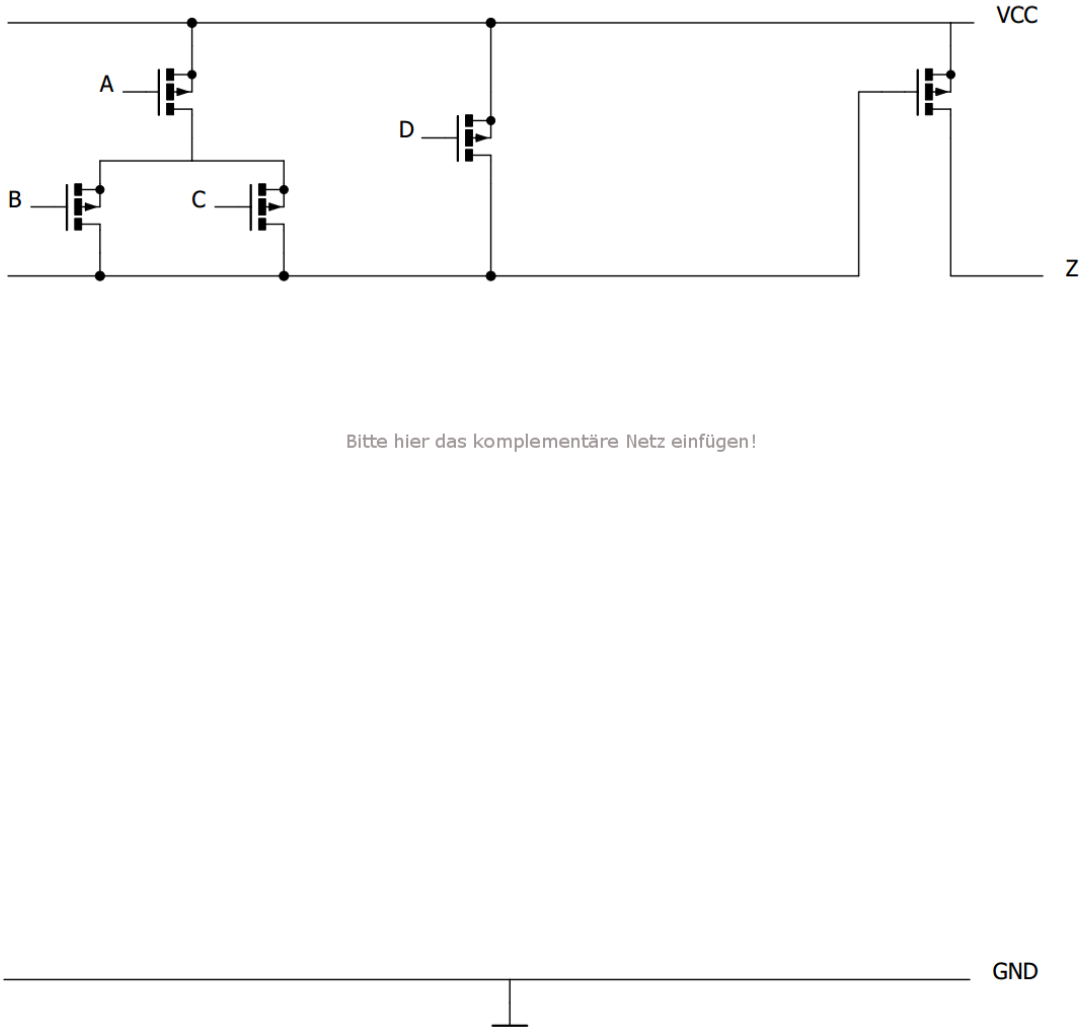


Abbildung 1

e) Bestimmen und ergänzen Sie die komplementäre Transistorschaltung im unteren Schaltungsteil von Abbildung 1 zwischen Ausgang *Z* und Masse *GND*.

Aufgabe 1.2: Steuerwerksentwurf *(12,5 Punkte)*

Gegeben sind das in Abbildung 1 dargestellte Operationswerk und das in Abbildung 2 gezeigte Steuerwerk.

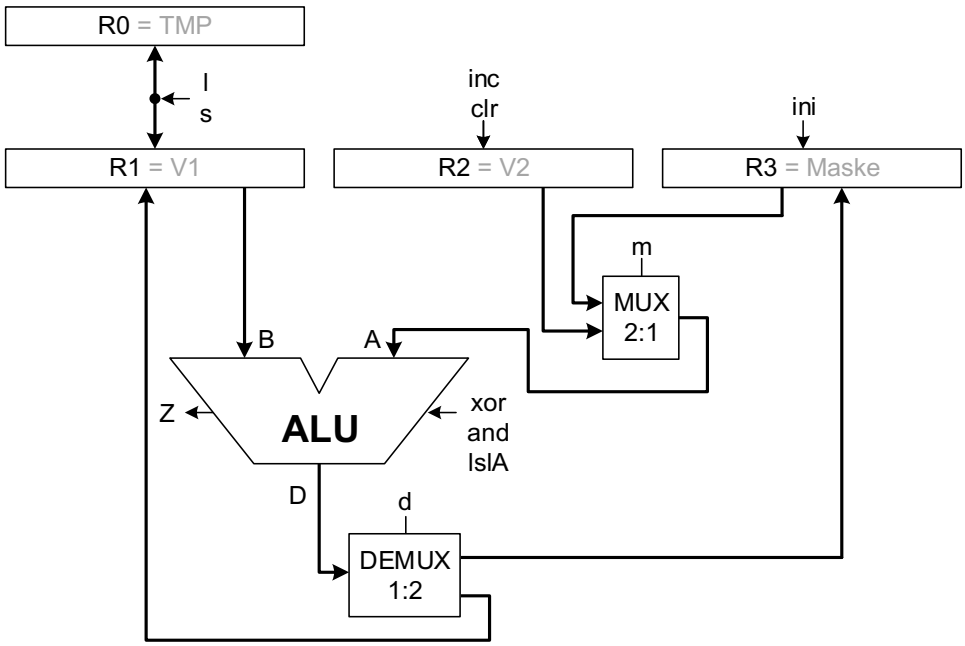


Abbildung 2: Operationswerk

Das gegebene Operationswerk basiert auf den vier 8-Bit Registern *R0* bis *R3* und einer 8-Bit ALU. Sowohl die Eingänge als auch der Ausgang der ALU und Register sind 8 Bit breit. Mithilfe vierer Steuersignale können Operationen auf der ALU ausgeführt werden. Das Steuersignal and und-verknüpft die an den Eingängen anliegenden Werte bitweise, das Signal *lslA* schiftet den am Eingang *A* anliegenden Wert unter Nachziehen einer 0 um eine Stelle nach links und das Steuersignal *xor* liefert die bitweise Anwendung des *xor* Operators auf die an den Eingängen anliegenden Werte. Ferner setzt die ALU das Signal *Z* für einen Takt auf High, wenn das Ergebnis der letzten Operation eine 0 war. Die Steuersignale *m* und *d* dienen zur Steuerung des MUX bzw. DEMUX. Das Signal *l* ermöglicht das Kopieren des Wertes von *R0* in das Register *R1* wohingegen das Signal *s* das Kopieren des Wertes aus *R1* in das Register *R0* realisiert. Das Register *R2* kann mithilfe des Steuersignals *clr* auf den Wert 0 zurückgesetzt und mit dem Signal *inc* um 1 inkrementiert werden. Das Signal *ini* ermöglicht die Initialisierung des Registers *R3* mit dem Wert 1.

Es soll ein Steuerwerk auf Basis eines Schieberegisters entworfen werden, welches den nachfolgend als Pseudocode gegebenen Algorithmus auf dem Operationswerk aus *Abbildung* ***2*** realsiert. Die Semikola dienen zur Trennung einzelner Takte.

Pseudocode	Registertransfer-operationen	Steuersignale
START: V1 = V1 xor V2;	_____	_____
TMP = V1, V2 = 0,	_____	_____
Maske = 1;	_____	_____
LOOP: V1 = V1 and Maske;	_____	_____
if V1 = 0 then	_____	
Maske = Maske << 1;	_____	_____
else		
Maske = Maske << 1,	_____	_____
V2 = V2 + 1;	_____	_____
fi		
if Maske != 0 then	_____	
V1 = TMP, goto Loop;	_____	_____
fi		

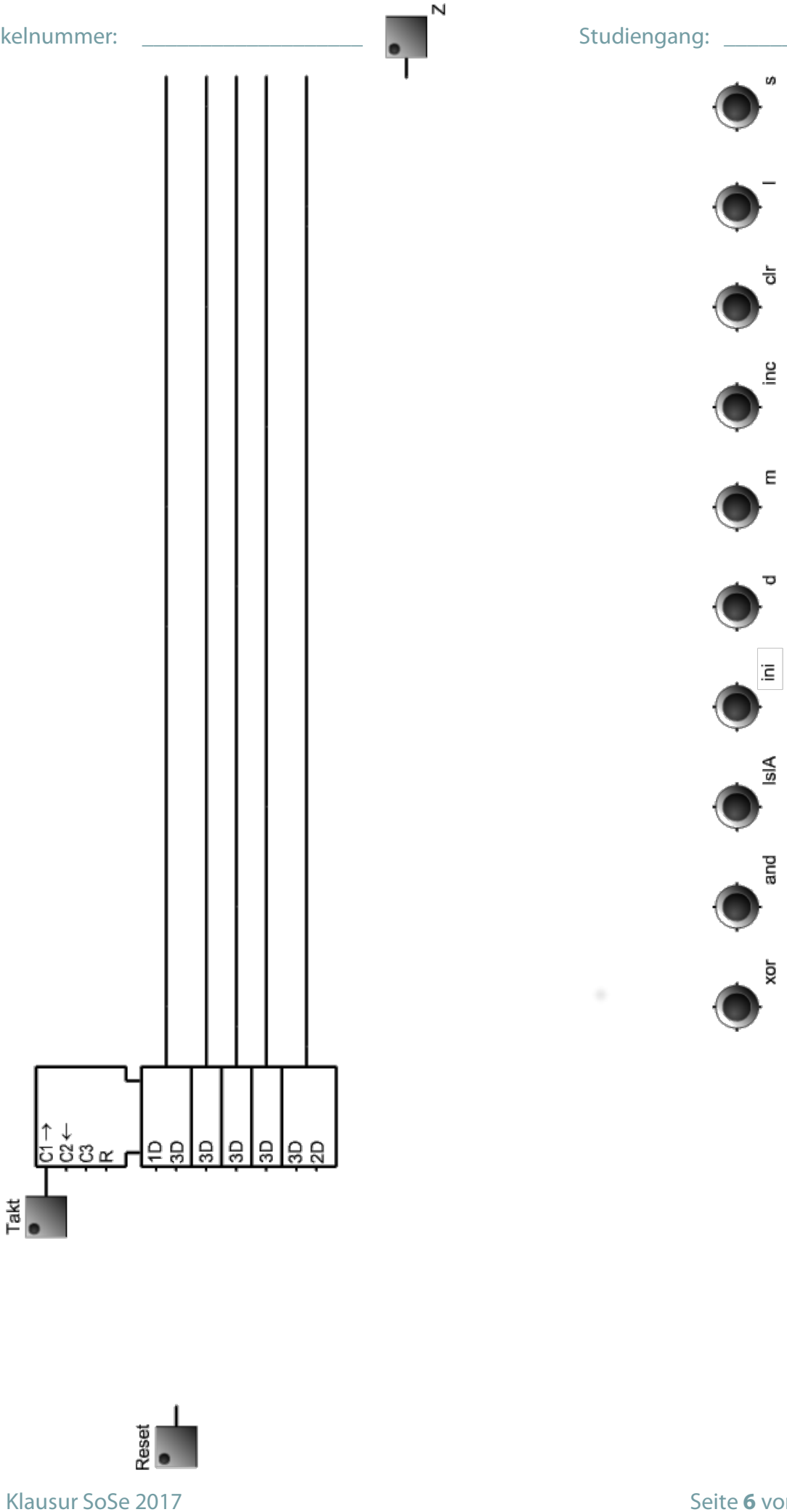
Tabelle 1: Algorithmus, Registertransferoperationen und Steuersignale

a) Geben Sie die auf dem Operationswerk auszuführenden Registertransferoperationen an, um die Berechnungen mit dessen Hilfe durchzuführen. Ergänzen Sie diese entsprechend in Tabelle 1. Beachten Sie dabei die in *Abbildung **2*** gegebene Zuordnung der Variablen zu den Registern.

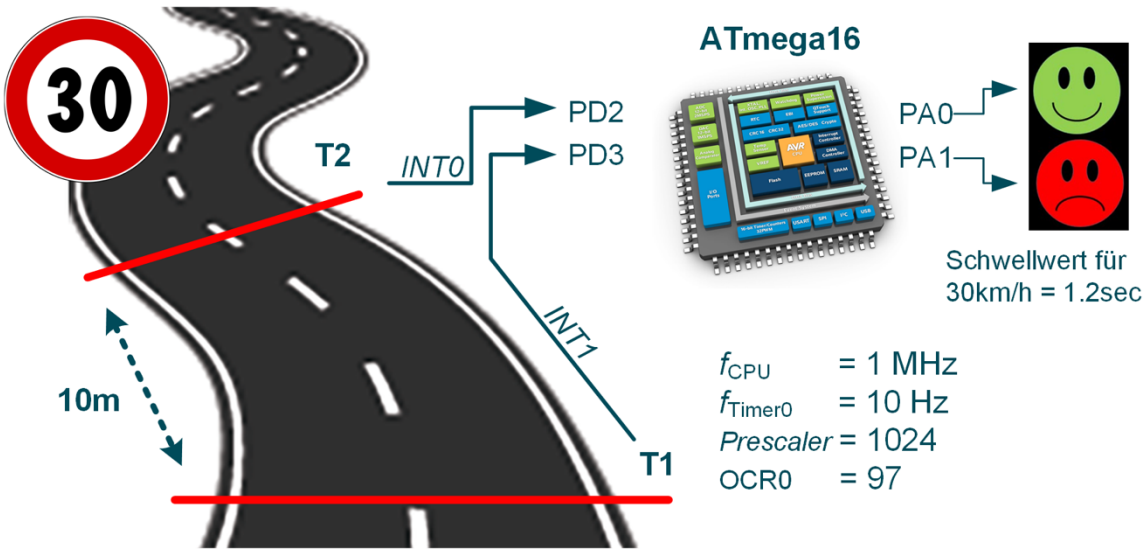
b) Ergänzen Sie nun Tabelle 1 um die notwendigen Steuersignale für die Realisierung des Algorithmus auf dem Operationswerk.

c) Welche Berechnung ist durch den Algorithmus beschrieben? Geben Sie eine kurze verbale Beschreibung in einem Satz an.

d) Entwerfen Sie nun anhand der Vorarbeiten aus a) und b) in LogiFlash-Notation ein Steuerwerk, welches den Algorithmus auf dem Operationswerk ausführt. Erweiter Sie hierfür die nachfolgende Schaltung. Ihnen Stehen UND- und ODER-Gatter sowie ein RS-Flipflop zur Verfügung. Gattereingänge dürfen negiert werden.


Z
s
l
clr
inc
m
d
ini
lslA
and
xor
C1→
C2←
C3
R
1D
3D
3D
3D
3D
3D
2D
Takt
Reset

Aufgabe 2: Assemblerprogrammierung (25 Punkte)



Mithilfe eines ATmega16 Mikrokontrollers soll die Steuerung einer Geschwindigkeitsüberwachung für eine Tempo 30 Zone implementiert werden. Dafür wurden unterhalb der Straße zwei Messbaken integriert, welche auf die Überfahrt eines Fahrzeugs reagieren. Das Prinzip ist dabei denkbar einfach: Das Fahrzeug passiert nacheinander die zwei Messbaken und das System misst die vergangene Zeit zwischen beiden Sensorreaktionen. Beträgt die Zeit mehr als 1.2*s*, wurde das Tempolimit eingehalten und ein grüner Smiley erscheint auf einer angeschlossen Ampel. Beträgt die vergangene Zeit weniger als 1.2*s*, liegt eine Geschwindigkeitsüberschreitung vor und ein roter Smiley erscheint.

Gehen Sie bei der Implementierung wie folgt vor: Die beiden Messbaken werden an die Pins `PD2` und `PD3` des ATmegas angeschlossen und generieren bei der Überfahrt einen externen Interrupt. Die Geschwindigkeitsmessung wird bei der Messschwelle `T1` mit der Reaktion des externen Interrupts `INT1` auf eine steigende Flanke gestartet. Darauffolgend startet der ATmega den internen `Timer0`, welcher in einem Register `time` die vergangene Zeit in Zehntelsekunden speichert. Die Ankunft des Fahrzeugs an der Messschwelle `T2` generiert dann den externen Interrupt `INT0`, welcher ebenfalls auf eine steigende Flanke reagiert. Dort wird die vergangene Zeit in ein Register `result` geschrieben, der Timer läuft allerdings im Hintergrund weiter.
Das Hauptprogramm läuft parallel in einer Endlosschleife und fragt periodisch den Wert im Register `result` ab. Ist dieser `0`, sollen die Anzeigen an den Pins `PA0` und `PA1` beide ausgeschaltet sein (logischer Wert `0`). Befindet sich in diesem Register jedoch ein Wert, welcher vom `INT0` hinein geschrieben worden ist, wird dieser ausgewertet. Ist der Wert größer oder gleich `12`, wird der grüne Smiley durch die Ausgabe einer `1` auf dem Pin `PA0` aktiviert. Ist der Wert kleiner `12`, wird der rote Smiley durch die Ausgabe einer `1` auf dem Pin `PA1` aktiviert. Die Anzeige soll 2*s* lang leuchten. Danach wird sie gelöscht und der `Timer0` beendet und zurücksetzt. Nutzen Sie für das Abwarten von 2*s* ebenfalls das Register `time`, welches vom `Timer0` weiterhin inkrementiert wird.

Aufgaben:

Die Bearbeitung der Aufgabe erfolgt in mehreren Unteraufgaben. Nutzen Sie die einzeln vorgegebenen Bereiche und schreiben Sie aussagekräftigen Assembler Code.

a. Beginnen Sie die Programmierung mit den einzelnen Registernamensdefinitionen. Sorgen Sie dafür, dass das Register `R16` unter dem Namen `time`, das Register `R17` unter dem Namen `result` angesprochen werden kann.

b. Initialisieren Sie die Interrupt-Vektor-Tabelle, sodass nach einem `RESET` zum Label `init`, nach einem *External Interrupt Request 0* zur `INT0_ISR`, nach einem *External Interrupt Request 1* zur `INT1_ISR` und nach *einem Timer/Counter 0 Compare Match* Interrupt zur `TIMER0_ISR` gesprungen wird.

c. Beginnen Sie nun unter dem Label `init` die Programmierung mit der Initialisierung des Stackpointers am Ende des `RAM`s und konfigurieren Sie die verwendeten EIN-/Ausgabeports.

d. Konfigurieren Sie den Timer/Counter`0` so, dass er beim Erreichen des Vergleichswerts 97 einen Interrupt auslöst. Verwenden Sie einen Prescaler von 1024 und aktivieren Sie den CTC Modus.

e. Initialisieren Sie im Folgenden die externen Interrupts `INT0` und `INT1`, welche ausgelöst werden, sobald eine steigende Flanke registriert wird. Aktivieren Sie dann die Interrupts global und springen sie anschließend zu `main`.

f. Implementieren Sie die `TIMER0_ISR` Interrupt Service Routine welche das interne Register `time` inkrementiert.

g. Implementieren Sie die `INT0_ISR` Interrupt Service Routine, welche den Wert von `time` auf 0 setzt und damit die Messung startet.

h. Implementieren Sie die `INT1_ISR` Interrupt Service Routine, welche den aktuellen Wert aus dem Register `time` in das Register `result` schreibt und damit die Auswertung initialisiert.

i. Implementieren Sie nun die beschriebene Grundfunktionalität unter dem Label `main`.

$\sum_{A2} =$ _____ Punkte