



UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TECHNISCHE INFORMATIK

Studienbegleitende Fachprüfung im Rahmen der
Bachelorprüfung

Sommersemester 2020

Lehrmodul:
Technische Grundlagen der Informatik 1

Prüfer: Dr.-Ing. Kristian Ehlers

15. Oktober 2020

Name: _____

Matrikelnummer: _____

Punkte:

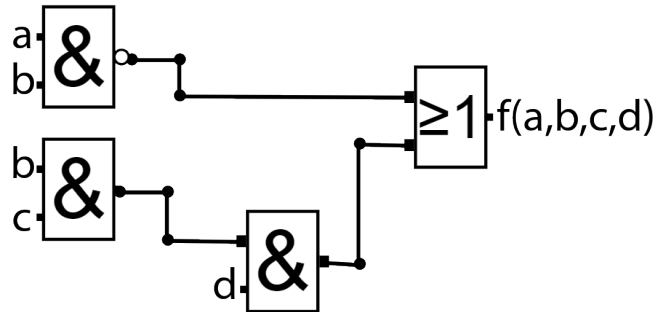
1	2	3	Σ

Matrikelnummer: _____

Studiengang: _____

Aufgabe 1: Technologien und Grundlagen

a) Gegeben sei folgende Schaltung:



Vervollständigen Sie die nachfolgende Wahrheitstabelle:

a	b	c	d	$f(a, b, c, d)$
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Geben Sie $f(a, b, c, d)$ nun in **konjunktiver kanonischer Normalform (KKN)** an.

Wie viele konjunktive kanonische Normalformen hat die Funktion f ? Begründen Sie Ihre Antwort.

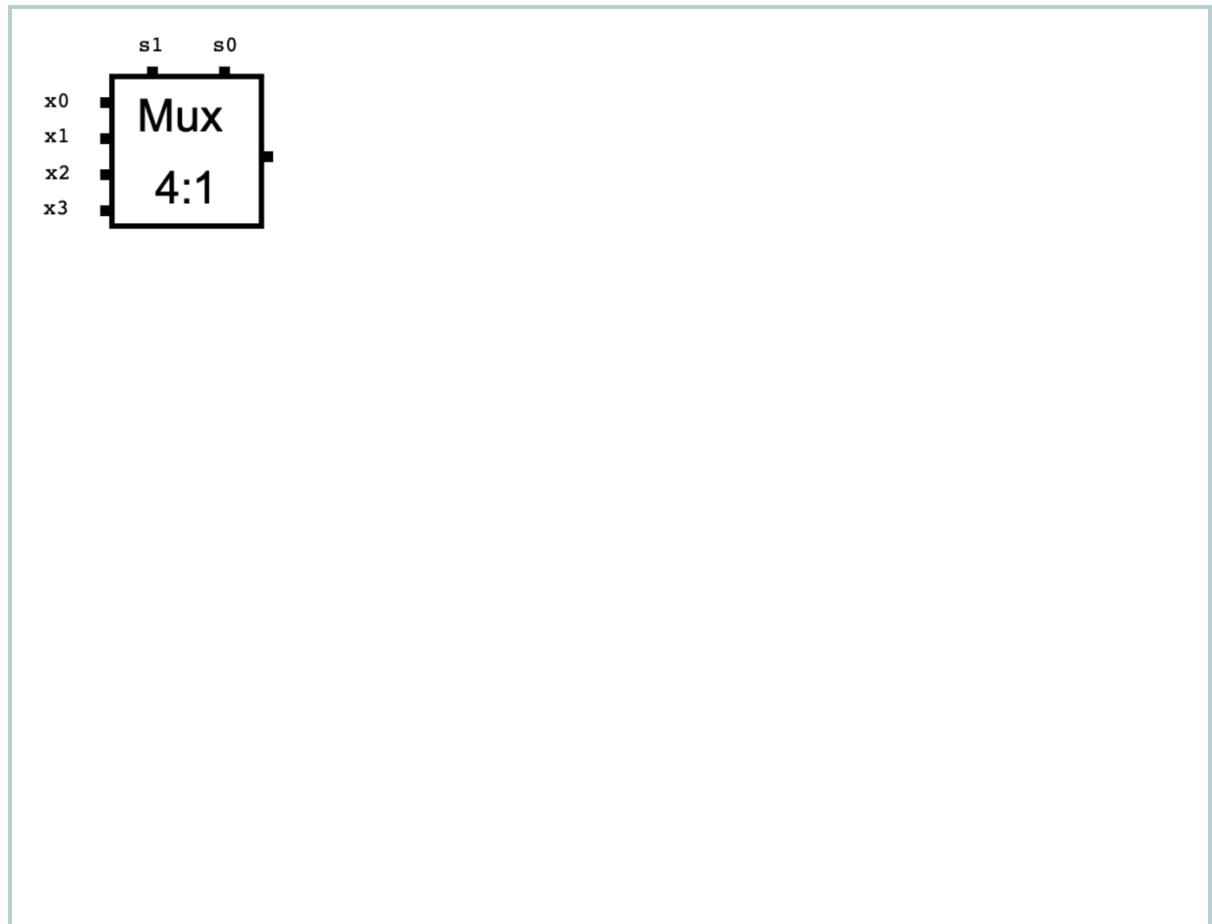
- b) Bestimmen Sie **alle disjunktiven Minimalformen (DMF)** von $g(a, b, c, d)$ an. Verwenden Sie für die Minimierung ein KV-Diagramm (ein Diagramm als Ersatz). Markieren Sie jeden zusammengefassten Term im KV-Diagramm und geben Sie letztlich die Minimalformen explizit unter Verwendung der Notation mit geschweiften Klammern an. Wie viele Minimalformen hat die Funktion?

$$g = (a + b + c + d)(a + b + c + \bar{d})(a + b + \bar{c} + \bar{d})(a + b + \bar{c} + d)(\bar{a} + b + c + d)(\bar{a} + \bar{b} + \bar{c} + \bar{d})$$

		a, b			
		00	01	11	10
c, d	00				
	01				
	11				
	10				

		a, b			
		00	01	11	10
c, d	00				
	01				
	11				
	10				

- c) Realisieren Sie den abgebildeten 4:1 MUX unter ausschließlicher Verwendung von 2:1 MUX-Bausteinen.



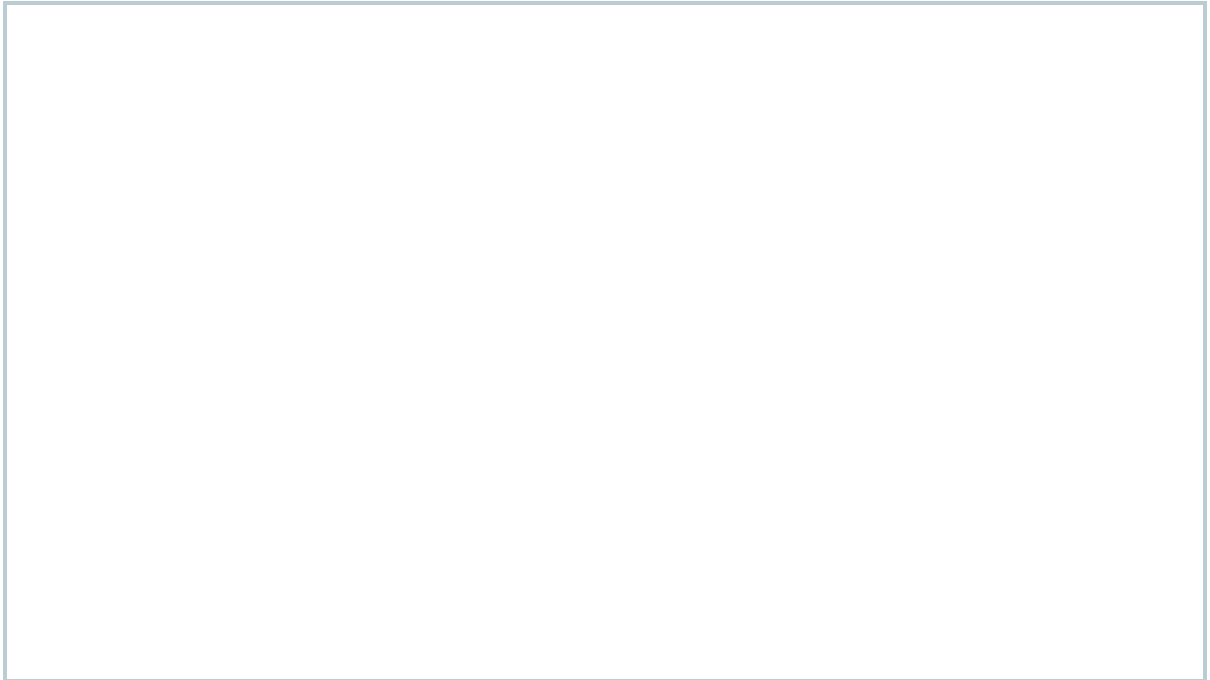
- d) Formen Sie $h(a, b, c)$ so um, dass sich die Schaltfunktion ausschließlich aus NAND-Ausdrücken über jeweils zwei Termen zusammensetzt. Das Resultat darf keine Negationen in Form eines Negations-Strichs enthalten. Verwenden Sie für die finale Darstellung die Notation: $(a|b)$.

$$h(a, b, c) = (a * \bar{b}) + (c \downarrow b)$$

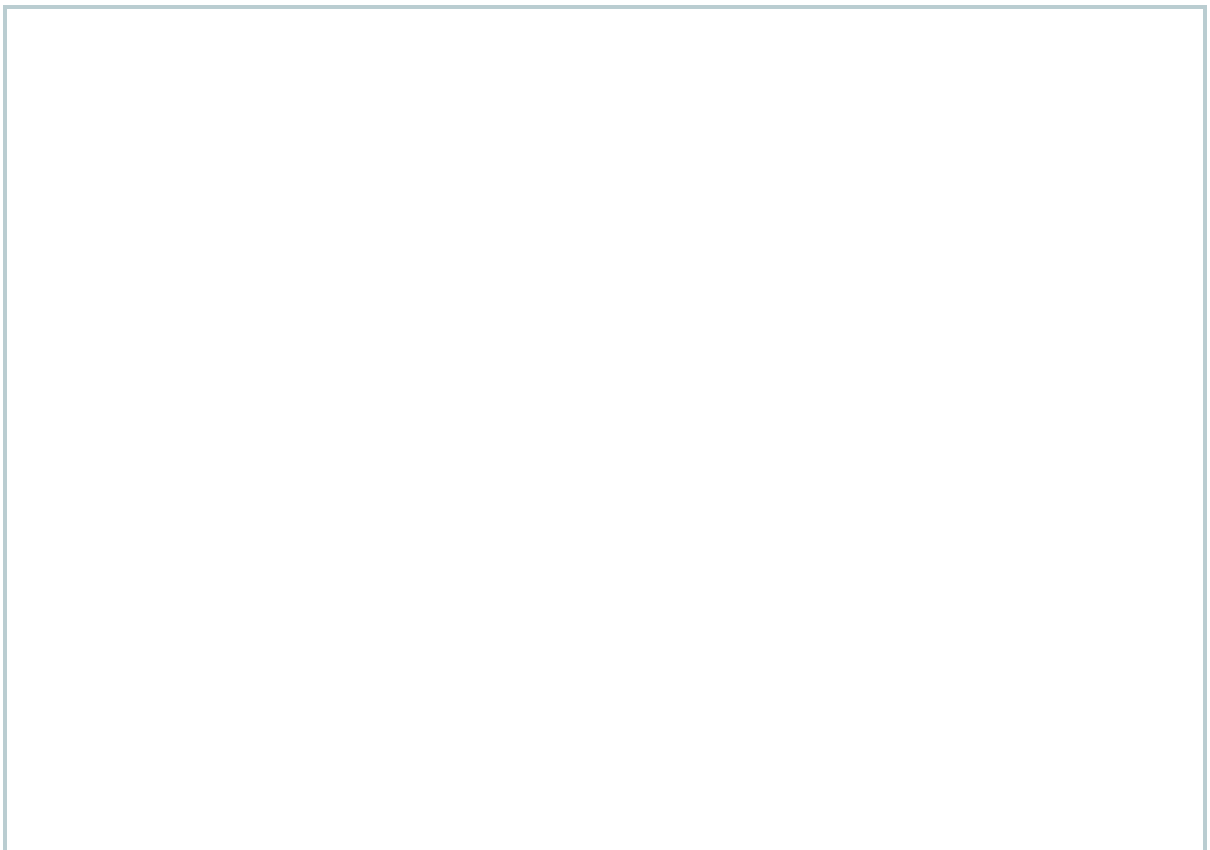
Matrikelnummer: _____

Studiengang: _____

Zeichnen Sie den Schaltplan der Schaltfunktion $h(a, b, c)$ unter Verwendung von Schaltsymbolen neuer DIN-Norm bestehend aus NAND-Gattern mit zwei Eingängen.



- e) Zeichnen Sie eine Schaltung in RTL-Technologie unter der Verwendung von npn-Transistoren, welche die Schaltfunktion $i(a, b, c) = (a + b) * c$ in NOR-Form realisiert. Geben Sie i explizit als Schaltfunktion in der NOR-Realisierung an.



- f) Geben Sie eine VHDL-Beschreibung (Entity und Architecture) der Funktion $h(a,b,c,d)$ an.

$$h(a,b,c,d) = (a + b) | ((b + c) \downarrow d)$$

$\sum_{A1} = \underline{\hspace{2cm}}$ Punkte

Aufgabe 2: Operations- und Steuerwerk

In dieser Aufgabe sollen ein Operations- und ein zugehöriges Steuerwerk auf Basis eines Schieberegisters entworfen werden, um den nachfolgenden, durch einen RT-Code gegebenen Algorithmus zu realisieren.

RT-Code	Takt	Kontrollsignale
declare register O, X(15:0), Y(15:0), CNT1(1:0), CNT2(1:0)		
declare bus BUS(15:0)		
INIT:		
CNT1 <- 0, CNT2 <- 0;	#Takt0	_____
READ:		
X <- BUS;	#Takt1	_____
CLR:		
Y <- 0;	#Takt2	_____
LOOP:		
O <- X(0), X(14:0) <- X(15:1), X(15) <- 0, Y <- Y + 1;	#Takt3	_____
if(O = 1) then Y <- Y - 1 fi;	#Takt4	_____
O <- Y(0), Y(14:0) <- Y(15:1), Y(15) <- Y(0), CNT1 <- CNT1 + 1;	#Takt5	_____
if(CNT1 <> 0) then goto LOOP fi;	#Takt6	_____
X <- X or Y, CNT2 <- CNT2 + 1;	#Takt7	_____
if (CNT2 = 0) then goto END else goto CLR fi;	#Takt8	_____
END: goto END;	#Takt9	_____

- a) Ergänzen Sie das Blockschaltbild des Operationswerks so, dass es die nachfolgend und durch den RT-Code beschriebenen Komponenten und Funktionalitäten aufweist. Das Operationswerk besitzt die zwei 16 Bit breiten Register X und Y, sowie die beiden 2 Bit Zählregister CNT1 und

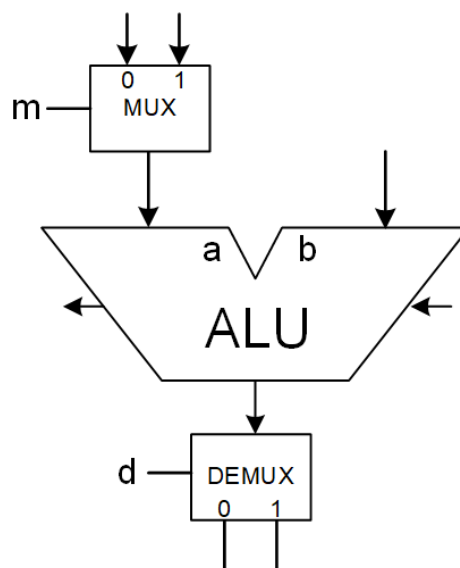
Matrikelnummer: _____

Studiengang: _____

CNT2, welche mit den Kontrollsignalen (KS) *incCNT1* bzw. *incCNT2* inkrementiert und mit *clrCNT1* bzw. *clrCNT2* auf Null gesetzt werden können. Zudem geben die Register mithilfe der Flags *zCNT1* bzw. *zCNT2* an, ob ihr Inhalt gerade Null ist ($zCNTx = 1$) oder nicht. Das Register Y kann über das KS *clrY* auf Null gesetzt werden, wohingegen *incY* den Inhalt inkrementiert und *decY* den Inhalt dekrementiert. Über das KS *inX* soll der Inhalt des 16 Bit breiten Busses *BUS* in das Register X geladen werden können. **Die Register X und Y sind keine Schieberegister!**

Das KS *or* sorgt dafür, dass die ALU die beiden an den Eingängen a und b anliegenden Werte bitweise miteinander verodert. Das KS *lsrA* realisiert, dass der am Eingang a anliegende Wert um ein Bit nach rechts unter Nachziehen einer Null geschiftet wird, wohingegen das KS *rsrA* dafür sorgt, dass die ALU einen Ringshift nach rechts mit am Eingang a anliegenden Wert durchführt. Zudem stellt die ALU das Flag O bereit, welches angibt, ob bei der letzten Operation ein Überlauf aufgetreten ist oder nicht bzw. ob eine 1 aus dem Register herausgeschiftet wurde ($O=1$) oder nicht.

BUS



- b) Ergänzen Sie die obige Tabelle mit den zu den einzelnen RT-Operationen korrespondierenden Steuersignalen.
- c) Welche Berechnung führt dieser Algorithmus durch bzw. welchen Wert enthält das Register X am Ende des Algorithmus? Geben Sie die semantische Bedeutung in einem Satz an.

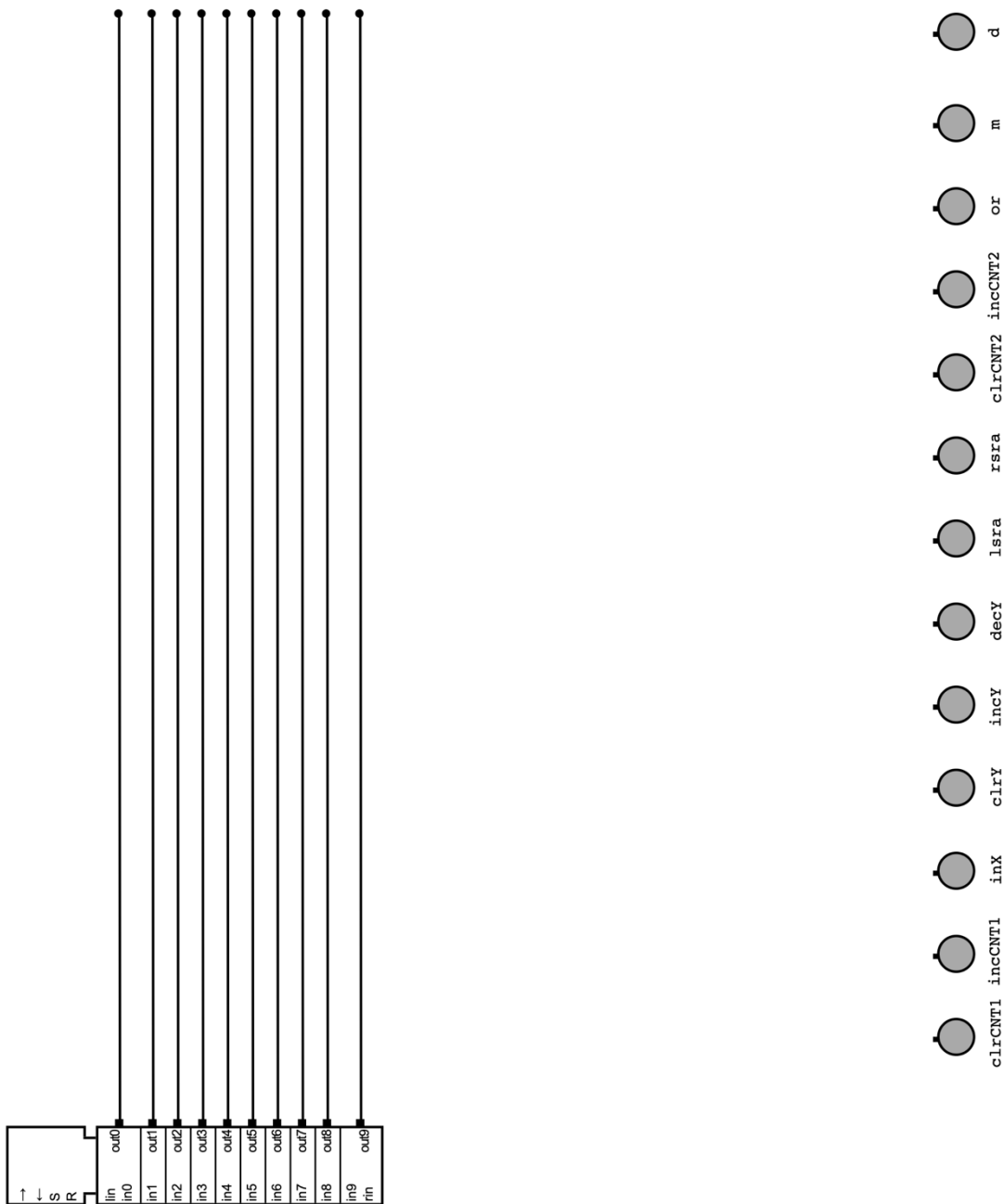
- d) Welchen Wert enthält das Register Y am Ende des Algorithmus?

- e) Welche Berechnung würde der Algorithmus durchführen, wenn der Counter CNT1 eine Breite von 3 Bit und CNT2 eine Breite von einem Bit bekommen würde?

- f) Könnte das Operationswerk auch weniger Register haben und dennoch die Berechnung ausführen? Begründen Sie Ihre Antwort und geben Sie kurz an, was gegebenenfalls geändert werden müsste.

- g) Durch welche Operation könnte *or* ersetzt werden ohne das Ergebnis des Algorithmus zu verändern.

- h) Entwerfen Sie ein Steuerwerk auf Basis eines Schieberegisters, welches den Algorithmus auf dem Operationswerk realisiert, indem Sie den nachfolgenden Entwurf vervollständigen. Bedenken Sie, dass nicht alle Kriterien aus dem RT-Code eins zu eins vom Operationswerk bereitgestellt werden. Positionieren Sie die Buttons für die Kriterien bei Bedarf nach Belieben.



Aufgabe 3: Assemblerprogrammierung

Rollosteuerung

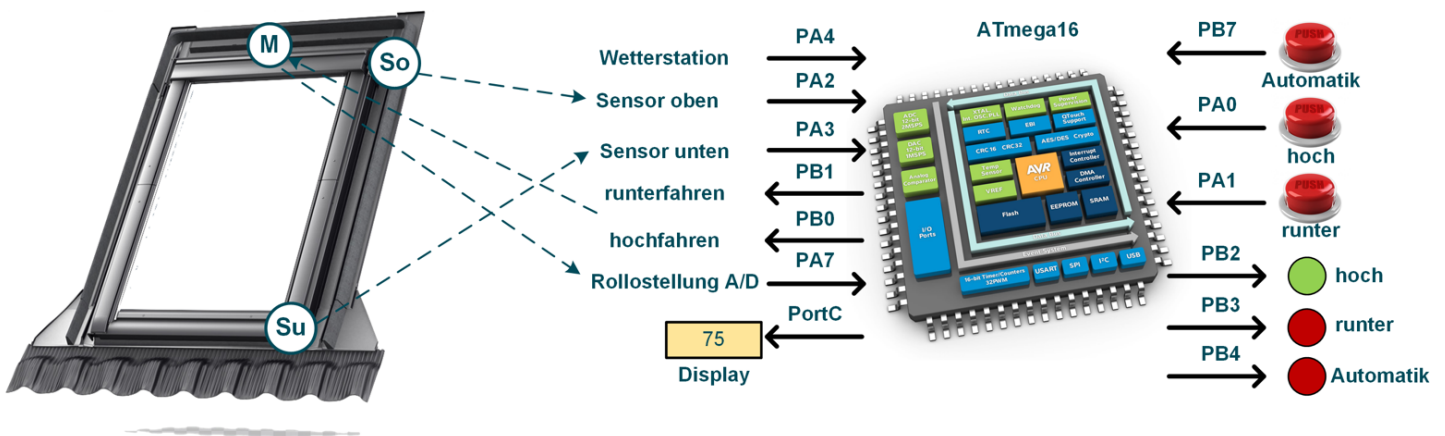


Abbildung 1: Beschaltung des ATmega16

In dieser Aufgabe sollen Sie eine Rollosteuerung mithilfe des ATmega16 realisieren. Wie Abbildung 1 zeigt, ist das Rollo mit verschiedenen Sensoren und Aktoren beschaltet.

Für das Bewegen des Rollos ist ein Motor zuständig, der über die Pins PB0 (1 = hochfahren) und PB1 (1 = runterfahren) angesteuert werden kann. Die Bewegungen werden mithilfe der an PA0 (1 = hochfahren) und PA1 (1 = runterfahren) angeschlossenen Taster initiiert. Sollte sich das Rollo bewegen, so wird die Richtung über die LEDs an PB2 (1 = hochfahren) und PB3 (1 = runterfahren) zusätzlich visualisiert. Die beiden Sensoren S_o und S_u zeigen an, ob sich das Rollo in der jeweiligen Endlage befindet ($S_x = 1$). Auf der an PortC angeschlossenen Anzeige wird jeweils die prozentuale Stellung des Rollos (0 = oben, 100 = unten) dargestellt. Diese wird mithilfe des an PA7 angeschlossenen analogen Sensors mit einer 10 Bit Genauigkeit bestimmt, wobei auch hier die 0 das geöffnete Rollo und die 1023 das geschlossene Rollo repräsentiert.

Die Funktionsweise kann wie folgt beschrieben werden. Wird ein Taster für weniger als 2 Sekunden gedrückt, so setzt sich das Rollo in Bewegung. Ein erneutes kurzes Drücken eines Tasters stoppt das Rollo wieder. Sollte der Taster länger als 2 Sekunden gedrückt werden, so setzt sich das Rollo in Bewegung und stoppt erst wieder, wenn der Taster losgelassen wird. Sollte das Rollo eine der Endlagen erreichen, so stoppt es und darf nicht weiter in diese Richtung bewegt werden.

Zusätzlich zum manuellen Betrieb gibt es den Automatikmodus, in dem das Rollo anhand des Signals einer Wetterstation herunterfährt, sollte die Sonne längere Zeit sehr stark scheinen ($PA4 = 1$). Es fährt wieder hoch, sollte die Sonne nicht mehr so stark scheinen ($PA4 = 0$). Das Rollo bewegt sich in beiden Fällen jeweils bis in die Endlage. Der Wechsel zwischen manuellem und automatischem Betrieb erfolgt über den Schalter an Pin PB7 (1 = Automatikmodus).

Aufgaben:

Die Bearbeitung der Aufgabe erfolgt in mehreren Unteraufgaben. Nutzen Sie die einzeln vorgegebenen Bereiche und schreiben Sie aussagekräftigen Assembler Code beziehungsweise kommentieren Sie gegebenenfalls den Code sinnvoll. Die Teilaufgaben sind sequentiell gestellt und sollen zusammen einen aneinanderkopierbaren Code ergeben. **Bedenken Sie in diesem Zusammenhang den eventuellen Wechsel zwischen Code- und Datensegment.**

- a) Sorgen Sie dafür, dass Ihnen die ATmega16-spezifischen Namensdefinitionen zur Verfügung stehen. Beginnen Sie anschließend die Programmierung mit den zusätzlichen Registernamensdefinitionen. Sorgen Sie dafür, dass das Register **R18** unter dem Namen **tmp1** und das Register **R19** unter dem Namen **tmp2** angesprochen werden können. Weitere von Ihnen eventuell genutzte Namen für einzelne Register müssen hier ebenfalls definiert werden.

- b) Reservieren Sie im RAM an der kleinstmöglichen Adresse unter dem Label **COUNTER** ein Byte, welches später für die Zeitmessung genutzt werden soll.

- c) Initialisieren Sie die Interrupt-Vektor-Tabelle, sodass nach einem **RESET** zum Label **INIT** und nach einem **Timer/Counter 0 Overflow Interrupt** zur **ISR_TIMER** gesprungen wird. Das Auftreten eines **AD Conversion Complete Interrupts** soll dazu führen, dass der ATmega in die **ISR_ADC** springt. Sorgen Sie in diesem Zusammenhang dafür, dass Ihr weiterer Quellcode erst nach der Interrupt-Vektor-Tabelle abgelegt wird.

- d) Konfigurieren Sie unter dem Label **INIT** die Ein- und Ausgabeports und initialisieren Sie den Stackpointer. Beachten Sie eventuelle Initialwerte für die Ausgänge. Sie dürfen **nicht** davon ausgehen, dass die Register standardmäßig mit dem Wert Null initialisiert sind. Konfigurieren Sie sämtliche Eingänge für den Betrieb im Tri-State Modus, schalten Sie die Motoren ab und die LEDs aus.

- e) Konfigurieren Sie den AD-Wandler, indem Sie ihn und seinen Interrupt aktivieren, eine rechtsbündige Speicherung einstellen und den Prescaler auf 64 konfigurieren. Die Versorgungsspannung soll als Referenzspannung genutzt werden. Bedenken Sie auch das Einstellen des richtigen Kanals.

- f) Initialisieren Sie den Timer/Counter 0 in der Art, dass er rund vier Mal in der Minute einen Overflow-Interrupt auslöst. Aktivieren Sie die Interrupts auch global.

- g) Nachfolgend ist das Unterprogramm **READ_ADC** teilweise gegeben. Ergänzen Sie den Code so, dass nach der AD/-Wandlung das 10 Bit breite Ergebnis in den Registern R17 und R16 zurückgegeben wird. R16 soll dabei die unteren 8 Bit des Ergebnisses beinhalten und R17 den Rest. Die ISR_ADC muss hier nicht implementiert werden.

```
READ_ADC:
    ldi    R16, (1 << SM0) | (1 << SE)
    out    MCUCR, R16

    SLEEP

    ; TODO Auslesen des Ergebnisses

    ret
```

- h) Implementieren Sie ein Unterprogramm **MOTOR_UP**, welches den Motor so ansteuert, dass das Rollo beginnt nach oben zu fahren, sollte es noch nicht in der Endstellung (komplett oben) angekommen sein. Denken Sie auch daran, dass die Fahrtrichtung bei Bewegung des Rollos mithilfe einer LED visualisiert wird.

Was müsste an dem Programm **MOTOR_UP** geändert werden, um daraus das Unterprogramm **MOTOR_DOWN** zu machen, welches dieselbe Funktionalität wie **MOTOR_UP** aufweist, nur das Rollo nach unten fahren lässt.

- i) Implementieren Sie die ISR_TIMER, die bei jedem Aufruf, den Wert der unter dem Label COUNTER im RAM zu finden ist, inkrementiert.

- j) Implementieren ein Unterprogramm **convert_AD_to_Percent**, welches den in den Registern R17 und R16 rechtsbündig gegebenen 10 Bit AD-Wert in die prozentuale Stellung des Rollos umrechnet. Ein AD-Wert von 0 entspricht dabei den 0% und alle Werte größer 990 entsprechen 100%.

- k) Realisieren Sie nun unter dem Label **MAIN** das Hauptverhalten des Systems. Verwenden Sie bei Bedarf alle realisierten und gegebene Unterprogramme. Das in einer **Endlosschleife** arbeitende Verhalten sei nachfolgend schrittweise skizziert
- Es erfolgt das Einlesen Stellung des Rollos und die Ausgabe in % auf **PortC**
 - Prüfen, ob der automatische oder manuelle Betrieb eingeschaltet ist.
 - Sollte der automatische Betrieb aktiviert sein, so wird dieser via LED (PB4) visualisiert und in Abhängigkeit vom Wert der Wetterstation entsprechend das Rollo hoch- oder runtergefahren.
 - Das Vorgehen beim manuellen Betrieb kann folgendermaßen beschrieben werden:
 - Visualisierung des manuellen Betriebes per LED (LED für Automatikbetrieb ausschalten)
 - Sollte kein Taster gedrückt sein wird wieder zum Beginn der **MAIN** gesprungen
 - Sollte ein Taster gedrückt werden, so muss die Zeit gemessen werden (**COUNTER** im RAM)
 - Wird die Taste weniger als 2 Sekunden lang gedrückt, so wird das Rollo in Bewegung gesetzt. Ein erneutes Drücken oder das Erreichen der Endlage sollen das Rollo wieder stoppen.
 - Wird die Taste länger als 2 Sekunden gedrückt, setzt sich das Rollo in Bewegung und stoppt, wenn die Taste losgelassen wird oder das Rollo seine Endlage erreicht hat.

Hinweis: Sie brauchen das Verhalten nur für das Hochfahren realisieren.

- l) Was muss an Ihrer Main angepasst werden um das Herunterfahren des Rollos zu realisieren? Eine kurze verbale Beschreibung genügt.

- m) Gegeben sei der nachfolgende Codeausschnitt zum Ablegen von Daten im ROM. Bitte geben Sie den resultierenden Inhalt des ROMs in Hexadezimaldarstellung an. Die Adressierung sei dabei mit Hilfe der Wortadressen vorzunehmen.

```
.cseg  
.org $FF  
DATA: .db 12, 235, 3, 1, 7, 5
```

Adresse	Inhalt in Hex

Geben Sie eine Folge von Assemblerbefehlen für den ATmega an, die in einer Schleife die Daten aus dem ROM aufsummiert und das Resultat in R16 hält. **Zum Abbruch der Schleife sei nur der Vergleich mit der Adresse des letzten Datums erlaubt!**

$\Sigma_{A3} = \underline{\hspace{2cm}}$ Punkte