

## Aufgabe 1: Technologien und Grundlagen (12,5 Punkte)

a) Gegeben sei folgende Wahrheitstabelle:

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

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

Realisieren Sie  $f(a, b, c, d)$  mithilfe eines 4:1 MUX-Bausteins, der Eingangsvariablen und beliebig vielen Invertern.



- b) Vereinfachen Sie  $g(a, b, c)$  algebraisch soweit wie möglich.

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

- c) Formen Sie  $h(a, b, c, d)$  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:  $(a \mid b)$ .

$$h(a, b, c, d) = (a + b) * (c \mid d)$$

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

- d) Realisieren Sie  $l(a, b, c) = a + (bc)$  unter Verwendung der CMOS-Technologie (Grundschialtung und Komplementärschaltung). Verwenden Sie folgende Symbole für n-Kanal-  und p-Kanal-CMOS-Transistoren .

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

Matrikelnummer: \_\_\_\_\_

Studiengang: \_\_\_\_\_

## Aufgabe 2: Steuerwerksentwurf

(12,5 Punkte)

Die Fletcher Prüfsumme dient in der digitalen Datenübertragung zur Erkennung von Fehlern. Eine Bitfolge wird in  $k$  Bit breite Blöcke unterteilt und jeder dieser so entstehenden  $n$  Blöcke als positive ganze Zahl interpretiert. Die Blöcke ergeben das Block-Array  $B$ . Ferner muss ein modulus Wert  $M$  festgelegt werden. Als Prüfsumme dienen die zwei Summen  $S1$  und  $S2$ , die durch den nachfolgenden Pseudocode definiert sind.

```
S1 := 0
S2 := 0

for i := 0 n do
    S1 := (S1 + B[i]) mod M
    S2 := (S2 + S1) mod M
end
```

Ziel dieser Aufgabe ist der Entwurf eines Operationswerks mit zugehörigem Steuerwerk in Form einer Zählersteuerung.

In dieser Aufgabe werden die als Parameter  $k = 4$  und  $M = 256$  genutzt. Die Berechnung der Summen ist durch das nachfolgende Registertransferprogramm gegeben, wobei das Register  $R$  die Bitfolge enthält, deren Prüfsumme berechnet werden soll.

```
declare register R(31:0), CNT(2:0), S1(3:0), S2(3:0), C

R <- 1413957937;

INIT:
    CNT <- 0, S1 <- 0, S2 <- 0, C <- 0;

LOOP:
    S1 <- S1 + R(3:0);

    S2 <- S2 + S1;

    R(27:0) <- R(31:4);

    C.CNT <- C.CNT + 1;

    if(C = 0) then
        goto LOOP
    fi;
```

Steuersignale d)

---

---

---

siehe Teilaufgabe e)

---

Matrikelnummer: \_\_\_\_\_

Studiengang: \_\_\_\_\_

- a) Welche Art von festverdrahteten Steuerwerken kennen Sie außer der Zählersteuerung noch?

- b) Die in  $R$  geladene initiale Bitfolge ist mit  $1413957937_{10}$  bzw.  $54474931_{16}$  als Hexadezimalwert gegeben und entspricht den ASCII-Codes der Sequenz „TGI1“. Welchen Inhalt in Hexadezimaldarstellung hat das Register  $R$  nach der Berechnung der Prüfsumme?

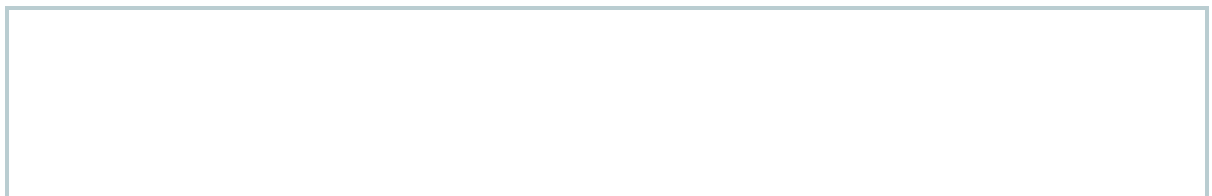
- c) Entwerfen Sie ein Blockschaltbild für das Operationswerk, welches durch den Registertransfercode beschrieben wird. Sie dürfen lediglich eine ALU verwenden, die wiederum nur addieren kann. Dazu muss das Steuersignal  $add$  aktiviert werden. Das Register  $R$  kann durch das Steuersignal  $lsrR$  um EIN Bit nach rechts ohne Nachziehen eines Wertes geschiftet werden. Das Register  $CNT$  wird mit Hilfe des Steuersignals  $inc$  inkrementiert. Sollte dabei ein Überlauf entstehen wird das Kriterium  $C$  (Carry) gesetzt. Zusätzlich zur ALU und den Registern stehen Ihnen ein 2:1 Mux und ein 1:2 Demux zur Verfügung, für deren Ansteuerung die Steuersignale  $m$  (Mux) beziehungsweise  $d$  (Demux) genutzt werden. Der Mux soll dazu dienen zwischen  $R(3:0)$  ( $m = 0$ ) und  $S2$  ( $m = 1$ ) als Eingabe für die Addition zu wechseln, wobei  $S1$  stets als Eingabe genutzt wird. Der Demux dient dazu das Ergebnis entweder in  $S1$  ( $d = 0$ ) oder  $S2$  ( $d = 1$ ) zu schreiben. Die Steuersignale  $clrCNT$ ,  $clrS1$  und  $clrS2$  ermöglichen das Setzen der entsprechenden Register auf den Wert 0.

**Hinweise:** Die Definition weiterer Steuersignale ist nicht gestattet. Behandeln Sie das Register  $C$  als Kriterium, welches vom Register  $CNT$  gesetzt beziehungsweise nach außen gegeben werden kann. Es gibt für  $C$  also keine Steuersignale.

- d) Ordnen Sie nun den einzelnen Registertransferoperation des Registertransferprogramms die in c) definierten Steuersignale zu, in dem Sie die aktiven Steuersignale neben die entsprechende Operation schreiben.
- e) Wie lässt sich die Registertransferoperation  $R(27:0) \leftarrow R(31:4)$  mit Hilfe des Operationswerks und der in c) durch die Steuersignale definierten, erlaubten Operationen realisieren? Geben Sie eine Abfolge von Registertransferoperationen und die zugehörigen Steuersignale an.



- f) Welchen Inhalt in Hexadezimaldarstellung hat das Register  $R$  nach der Berechnung der Prüfsumme und der Verwendung Ihres Registertransfercodes aus e) anstelle der Operation  $R(27:0) \leftarrow R(31:4)$ ?



- g) Entwerfen Sie nun das Steuerwerk für die Realisierung der Berechnung der Prüfsumme auf dem Operationswerk gemäß des *gegebenen* Registertransferprogramms indem Sie die nachfolgende Zählersteuerung ergänzen. Verwenden Sie das Steuersignal  $lstr4R$ , welches die Registertransferoperation  $R(27:0) \leftarrow R(31:4)$  auf  $R$  auslöst.





- h) Wäre eine Realisierung unter Verwendung von *lsrR* und Ihren Registertransfercodes aus e) anstelle von *lsr4R* und  $R(27:0) \leftarrow R(31:4)$  problemlos mit dem gegebenen Zähler möglich? Begründen Sie Ihre Antwort kurz.

$\Sigma_{A2} = \underline{\hspace{2cm}}$  Punkte

Matrikelnummer: \_\_\_\_\_

Studiengang: \_\_\_\_\_

## Aufgabe 3: Assemblerprogrammierung (25 Punkte)

### Startprozedur der Formel 1

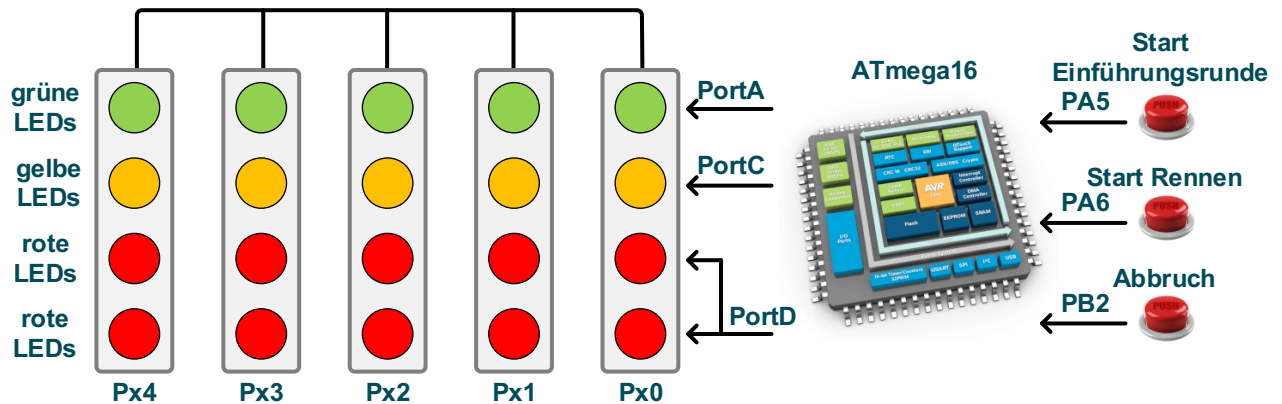


Abbildung 1: Aufbau der Startanlage

Ziel dieser Aufgabe ist die Implementierung einer leicht abgeänderten Variante der Startprozedur der Formel 1 in Assembler für den aus der Übung bekannten Mikrocontroller ATmega16, der gemäß Abbildung 1 mit der Ampelanlage und drei Knöpfen verschaltet ist.

Der Start wird in zwei Phasen unterteilt, deren Abläufe in Form sequentiell aufeinanderfolgender Ampelbeschaltungen spaltenweise in Tabelle 1 aufgeführt sind. Eine Viertelstunde vor dem geplanten Rennstart wird die Startphase für die Einführungsrunde manuell durch die Betätigung des an PA5 angeschlossenen Knopfes gestartet. Im Anschluss an die Einführungsrunde erfolgt der eigentliche Start des Rennens, der wiederum durch die Betätigung des an PA6 angeschlossenen Knopfes in Gang gesetzt wird. Mit Hilfe des an PB2 angeschlossenen Knopfes kann die Startprozedur jederzeit abgebrochen werden, was den Fahrern visuell durch das Leuchten aller gelben LEDs angezeigt wird.

Für jede Ampelphase werden die Anzeigedauer und das Ampelmuster mit Hilfe des nachfolgenden Assemblercodes aus Tabelle 2 im ROM abgelegt. Zu einem Muster gehören vier 8 Bit Werte: die Anzeigedauer des Musters sowie die Beschaltungen der einzelnen LEDs (grün, gelb und rot), die als Binärzahl interpretiert werden und deren Bitstellen direkt zu den Pins eines Ports korrespondieren. Wird eine 1 an einem Pin ausgegeben, leuchtet die angeschlossene LED und bei einer 0 ist sie ausgeschaltet. Das MSB der Zeitangabe kodiert, ob es sich bei dem durch die restlichen 7 Bit dargestellten Zahlenwert um eine Minutenangabe (MSB = 1) oder eine Sekundenangabe (MSB = 0) handelt.

Tabelle 1: Ampelphasen der Startprozedur

Wie lange vor dem Start der Einführungs- runde	Ampelbeschriftung Start der Einführungs- runde	Wie lange vor dem Start des Rennens	Ampelbeschriftung Start des Rennens
15 min		5 sek	
5 min		4 sek	
3 min		3 sek	
1 min		2 sek	
15 sek		1 sek	
0 sek		0 sek	
Abbruch			

**Hinweis:** Die LEDs einer „Zeile“ haben stets die gleiche Farbe. In der Tabelle weiß gefüllte LEDs leuchten wohingegen schwarz gefüllte LEDs nicht leuchten.

grüne LEDs

gelbe LEDs

rote LEDs

rote LEDs

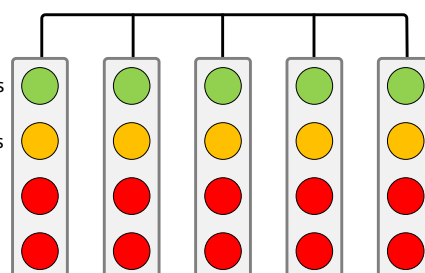


Tabelle 2: Assemblercode zum Ablegen der Ampelmuster im ROM

```

; Ablegen direkt nach der Interrupt Vektor Tabelle

PATTERNS_INTRODUCTION: ; Einföhrungsrunde
                        ; Zeit    grün  gelb  rot
                        ;          PortA PortC PortD
                        .db 0x8A,    0,    0, 0x1F ; 10 Minuten und alle roten LEDs an
                        .db 0x82,    0,    0, 0x0F ; 2 Minuten und nur die vier rechten Spalten
                                                ; mit roten LEDs an
                        .db 0x82,    0,    0, 0x07 ; 2 Minuten und nur die drei rechten Spalten
                                                ; mit roten LEDs an
                        .db 0x2D,    0,    0, 0x03 ; 45 Sekunden und nur die zwei rechten Spal-
                                                ; ten mit roten LEDs an
                        .db 0x0E,    0,    0, 0x01 ; 15 Sekunden und nur die rechte Spalte mit
                                                ; roten LEDs an
                        .db 0x0A, 0x1F,    0,    0 ; 10 Sekunden und alle grünen LEDs an
PATTERNS_RACE: ; Rennstart
                        .db 0x02,    0,    0, 0x10
                        .db 0x01,    0,    0, 0x18
                        .db 0x01,    0,    0, 0x1C
                        .db 0x01,    0,    0, 0x1E
                        .db 0x01,    0,    0, 0x1F
                        .db 0x01,    0,    0, 0x00
PATTERN_ABORT:
                        .db 0x00,    0, 0x1F, 0x00

```

## 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.

- Ergänzen Sie den Assemblercode aus Tabelle 2 in der Art, dass das Ablegen der Daten direkt im Anschluss an die Interrupt Vektor Tabelle erfolgt.

- b) Beginnen Sie die Programmierung mit den einzelnen Registernamensdefinitionen. Sorgen Sie dafür, dass das Register **R17** unter dem Namen **Abort**, das Register **R18** unter dem Namen **Seconds** und das Register **R19** unter dem Namen **Minutes** angesprochen werden können. Weitere von Ihnen eventuell genutzte Namen für einzelne Register müssen hier ebenfalls definiert werden.

- c) Initialisieren Sie die Interrupt-Vektor-Tabelle, sodass nach einem **RESET** zum Label **INIT**, nach dem **External Interrupt Request 2** zur **ISR\_INT2** und nach einem **Timer/Counter 1 Compare Match Interrupt** zur **ISR\_TIMER1** gesprungen wird.

- d) Konfigurieren Sie unter dem Label **INIT** die Ein- und Ausgabeports und initialisieren Sie den Stackpointer. Beachten Sie eventuelle Initialwerte für die Ausgänge. Initialisieren Sie das Register **Abort**, welches angibt, ob der Start abgebrochen wurde (**Abort** = 0xFF) oder nicht (**Abort** = 0).

- e) Konfigurieren Sie den **externen** Interrupt INT2 in der Art, dass er bei einer durch den an PB2 angeschlossenen Knopf hervorgerufenen **fallenden Flanke** ausgelöst wird.

- f) Die Messung der Zeitdauer des Anzeigens eines Ampelmusters erfolgt mit Hilfe des **Timer/Counter 1**. Konfigurieren Sie ihn so, dass er **einmal pro Sekunde** einen Compare Match Interrupt auslöst. Verwenden Sie den **CTC-Modus** und nutzen Sie einen **Prescaler** von **256**. Berechnen Sie den entsprechenden Vergleichswert. Aktivieren Sie den Interrupt des Timers und zudem alle Interrupts global.

- g) Implementieren Sie die Interrupt-Service-Routine **ISR\_INT2** die den Wert im Register **Abort** auf 0xFF setzt.



- h) Implementieren Sie die **ISR\_TIMER1** Interrupt Service Routine, welche den aktuellen Wert des Registers **Seconds** inkrementiert und beim Erreichen eines Registerwertes von 60, diesen auf 0 setzt und den Wert des Registers **Minutes** inkrementiert.

- i) Implementieren Sie nun das Unterprogramm **PHASE**, welches den Ablauf einer Startphase (Einführungsrunde oder Rennen) regelt. Es erwartet im **Z** Register die Adresse des ersten Musters der Phase und lädt die entsprechenden Informationen eines einzelnen Musters dieser Phase sequentiell aus dem ROM (jeweils die vier Werte: Zeit, grüne LEDs, gelbe LEDs, rote LEDs) und zeigt das Muster für die angegebene Dauer an, bevor es das nächste Muster lädt und dieses anzeigt. Wurden alle Muster der Startphase angezeigt, wird das Unterprogramm verlassen und das letzte Muster weiter angezeigt. Beachten Sie, dass es jederzeit zu einem Startabbruch kommen kann. Im Falle eines Abbruchs soll das Unterprogramm verlassen werden. Realisieren Sie diese Funktionalität mit Hilfe von Polling des Wertes des Registers **Abort**, während des Wartens für die Anzeigedauer eines Musters. Das Warten selbst kann mit Hilfe der Register **Seconds** und **Minutes** erfolgen. Zu Beginn des Wartens werden diese auf 0 gesetzt und anschließend entsprechend auf die verstrichene Zeit geprüft.  
Hinweis: Jede Phase besteht aus sechs Mustern.



- j) Implementieren Sie das Grundverhalten unter dem Label **MAIN**. Warten Sie, bis der an PA5 angeschlossene Knopf betätigt wird ( $\text{PinA5} = 0$ ). Laden Sie anschließend die Adresse des ersten Musters der Einführungsrunde in das Z Register und rufen Sie das Unterprogramm PHASE auf. Testen Sie nach der Abarbeitung des Unterprogramms, ob der Start abgebrochen wurde und springen Sie gegebenenfalls zum Label `START_ABORTED`, unter welchem das Muster für den Abbruch aus dem ROM geladen werden soll, die Ampeln entsprechend beschalten werden müssen und im Anschluss in einer Endlosschleife verharren werden soll. Sollte der Start nicht abgebrochen worden sein, so warten Sie, bis der Knopf an PA6 betätigt wird ( $\text{PinA6} = 0$ ) und starten Sie anschließend die Prozedur für den Start des Rennens. Im Anschluss an den Rennstart soll der Atmega in einer Endlosschleife verharren. Beachten Sie, dass es auch während des Wartens auf die Betätigung des Knopfes für den Rennstart zum Abbruch des gesamten Startvorgangs kommen kann.

$\Sigma_{A3} =$  \_\_\_\_\_ Punkte