HÖHERE TECHNISCHE BUNDESLEHRANSTALT

HOLLABRUNN

Höhere Abteilung für Elektronik – Technische Informatik

Klasse / Jahrgang: 4BHEL	Übungsbetreuer Prof. Wihsböck
Übungsnummer: ASM_CM3/1	Übungstitel: Personenzähleinrichtung
Datum der Vorfhrung: 17.01.2020	Gruppe: Platajs Martin, Roll Maximilian
Datum der Abgabe: 22.01.2020	Unterschrift:

Beurteilungskriterien

Programm:	**Punkte**
Programm Demonstration	
Erklärung Programmfunktionalität	
Protokoll:	**Punkte**
Pflichtenheft (Beschreibung Aufgabenstellung)	
Beschreibung SW Design (Flussdiagramm, Blockschaltbild,..)	
Dokumentation Programmcode	
Testplan (Beschreibung Testfälle)	
Kommentare / Bemerkungen	
Summe Punkte	

Note: _____

Inhaltsverzeichnis

1 **Angabe**

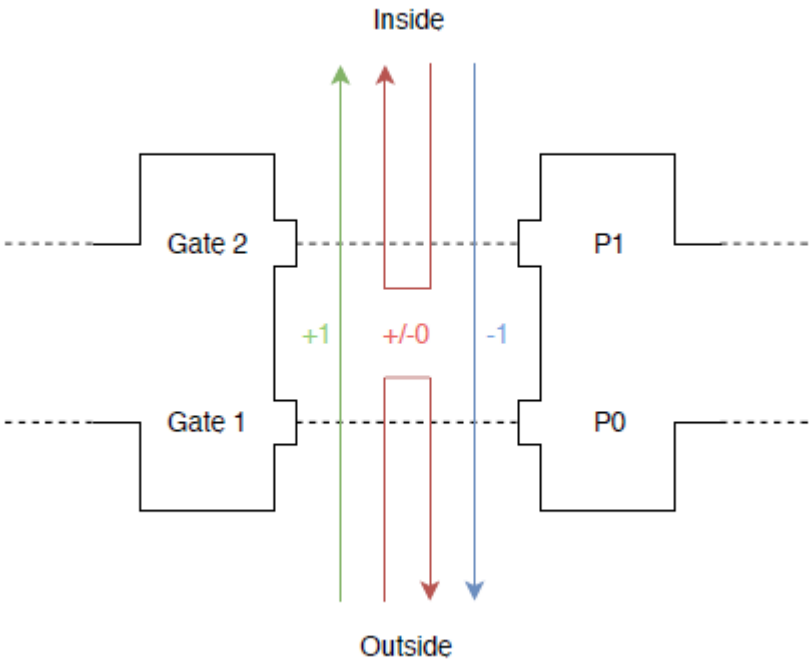
2 Anforderungen

Mithilfe der DIP-Schalter sollen zwei Lichtschranken simuliert werden, welche hochzählen bzw. runterzählen, wenn Personen durchgehen. Auf den LEDs soll binär dargestellt werden wieviel Personen nun im „Bereich“ befinden. Dabei sind folgende Sonderfälle zu beachten:

- Wenn das 2. Gate innerhalb 10s nicht ausgelöst wird, soll der Zählvorgang verworfen werden.
- Wenn ein Gate zweimal hintereinander ausgelöst wird, soll der Zählvorgang verworfen werden.
- Wenn beide Gates gleichzeitig ausgelöst werden soll der Piezo ausgelöst werden.

Außerdem soll über die V24-Schnittstelle mittels UART2 ein Begrüßungstext und alle Zustandsänderungen ausgegeben werden. Die Baudrate beträgt 4800 mit 8 Datenbits und einem Stoppbit.

2.1 Visualisierung



2.2 LED- und Schalterbelegung

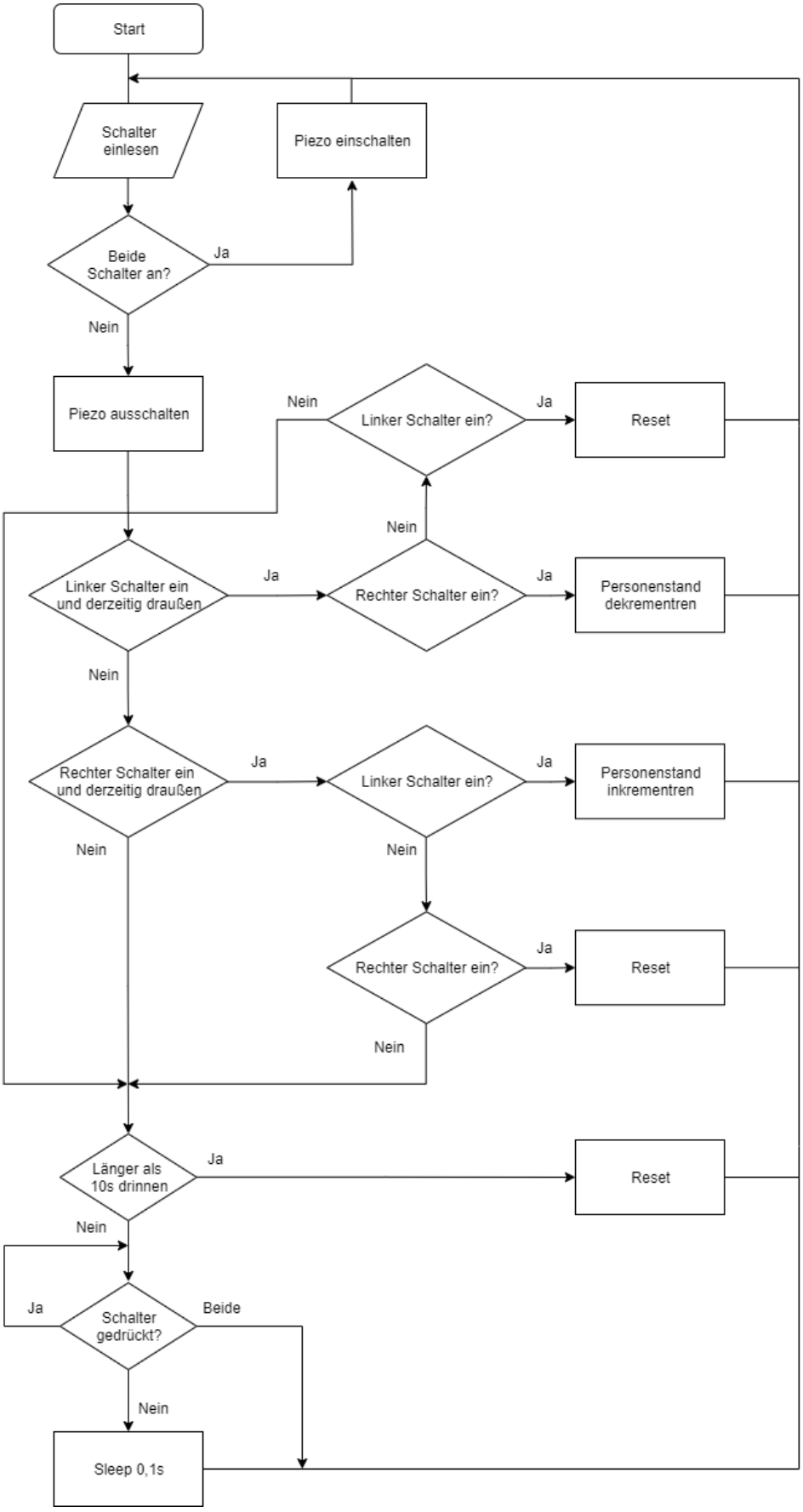
SCHALTER							
PA7	*PA6*	*PA5*	*PA4*	*PA3*	*PA2*	*PA1*	*PA0*
S7	S6	S5	S4	S3	S2	S1	S0
Gate 2	-	-	-	-	-	-	**Gate 1**

LEDs							
PB15	*PB14*	*PB13*	*PB12*	*PB11*	*PB10*	*PB9*	*PB8*
LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
Counter	**Counter**	**Counter**	**Counter**	**Counter**	**Counter**	**Counter**	**Counter**

2.3 Registerbelegung

Register	Verwendungszweck
R0	Allgemeines
R1	Allgemeines
R2	Allgemeines
R3	unused
R4	
R5	
R6	
R7	
R8	uart
R9	counter
R10	State
R11	time
R12	direction
R13	Stack-Pointer
R14	Link-Register
R15	Program-Counter

3 Softwaredesign



4 Programlisting

```
;*******************************************************************************
;* C) Copyright HTL - HOLLABRUNN  2020                                         *
;*                                                                             *
;* File Name:   pplcnt.s                                                       *
;* Autor:       Maximilian Roll & Martin Platajs                               *
;* Version:     V1.1                                                           *
;* Date:        14.01.2020                                                     *
;* Patch:       21.01.2020                                                     *
;*              Fixed overflow error while checkout                            *
;* Description: Personenzaehleinrichtung mithilfe von Lichtschranken           *
;*******************************************************************************

                AREA PPLCNT, CODE, READONLY
                INCLUDE STM32_F103RB_MEM_MAP.INC
                EXPORT __main
```

4.1 Definitionen

```
;****************************** Definitionen ******************************
;R8 ...uart
;R9 ...counter
;R10...state
;R11...time
;R12...direction(00...outside/01...goingIn/10...goingOut)

;Definition Gates:
Gate1           EQU     PERIPH_BB_BASE+(GPIOA_IDR - PERIPH_BASE)*0x20+0*4
Gate2           EQU     PERIPH_BB_BASE+(GPIOA_IDR - PERIPH_BASE)*0x20+7*4

;Definition LEDs:
LED0            EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+8*4
LED1            EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+9*4
LED2            EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+10*4
LED3            EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+11*4
LED4            EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+12*4
LED5            EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+13*4
LED6            EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+14*4
LED7            EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+15*4

;Definition Piezo:
PIEZO           EQU     PERIPH_BB_BASE+(GPIOB_ODR - PERIPH_BASE) * 0x20+0*4

baudrate_const  DCD     8000000
```

```
Beg_txt         DCB "Peoplecounter - V1\r\n",0  ;Begruessungstext
msg_in          DCB "Person ist reingegangen\r\n",0
msg_out         DCB "Person ist rausgegangen\r\n",0
msg_gate        DCB "Person in Schleuse\r\n",0
msg_time        DCB "Fehlausloesung\r\n",0
msg_dbbl        DCB "Doppelausloesung\r\n",0
```

4.2 Hauptprogramm

```
;*******************************************************************************
;*                        M A I N  P r o g r a m m:                            *
;*******************************************************************************

__main          PROC
                BL      init_ports
                BL      init_uart
                BL      init_variables
                LDR     R8, =Beg_txt
                BL      uart_put_string

main_loop       LDR R0, =GPIOA_IDR ;Alle Schalter einlesen
                LDR R1, [R0]
                AND R10, R1, #0x81 ;Schalter 7 & 0 abmaskieren

                CMP R10, #0x81
                BNE _no_error

                LDR R0, =PIEZO
                MOV R1, #0x1
                STR R1, [R0]
                BL alarm_wait
                MOV R1, #0x0
                STR R1, [R0]
                BL alarm_wait

_no_error       CMP R10, #0x81
                BEQ main_loop

                BL check_double

                BL check_in

                BL check_out

                BL check_timeout

                BL inc_time
```

```
_switch_hold    LDR R0, =GPIOA_IDR
                LDR R1, [R0]
                AND R10, R1, #0x81 ;Schalter 7 & 0 abmaskieren
                CMP R10, #0x01
                BEQ _switch_hold
                CMP R10, #0x80
                BEQ _switch_hold
                CMP R10, #0x81
                BEQ main_loop

                BL wait_100ms

                B main_loop

                ENDP
```

4.3 init_ports

```
;*******************************************************************************
;*            U N T E R P R O G R A M M:    init_ports                         *
;*                                                                             *
;* Aufgabe:   Initialisiert Portleitungen fuer LED / Schalterplatine          *
;* Input:     keine                                                           *
;* return:    keine                                                           *
;*******************************************************************************

init_ports      PROC
                push {R0-R7,LR}      ;save link register to Stack

                MOV R2, #0x4            ; enable clock for GPIOA (APB2
                LDR R1, =RCC_APB2ENR    ; Peripheral clock enable register)
                LDR R0, [R1]
                ORR R0, R0, R2
                STR R0, [R1]

                MOV R2, #0x8            ; enable clock for GPIOB (APB2
                LDR R1, =RCC_APB2ENR    ; Peripheral clock enable register)
                LDR R0, [R1]
                ORR R0, R0, R2
                STR R0, [R1]

                LDR R1, =GPIOA_CRL      ; set Port Pins PA0 to Pull Up/Down In
                LDR R0, [R1]            ; put mode (50MHz) - Schalter S0
                LDR R2, =0x0FFFFFF0
                AND R0, R0, R2
                LDR R2, =0x80000008
                ORR R0, R0, R2
                STR R0, [R1]

                LDR R1, =GPIOA_ODR      ; GPIOA Output Register Bit 0 auf "1"
                LDR R0, [R1]            ; sodass Input Pull Up aktiviert ist!!
                LDR R2, =0x81
                ORR R0, R0, R2
                STR R0, [R1]
```

```
LDR R1, =GPIOB_CRH      ; set Port Pin PB8 to Push Pull Output
LDR R0, [R1]            ; Mode (50MHz) - LED0
LDR R2, =0x00000000
AND R0, R0, R2
LDR R2, =0x33333333
ORR R0, R0, R2
STR R0, [R1]

LDR R1, =GPIOB_CRL      ; GPIOB PB0 (PIEZO) Push Pull Output
LDR R0, [R1]
LDR R2, =0xFFFFFFF0
AND R0, R0, R2
MOV R2, #0x03
ORR R0, R0, R2
STR R0, [R1]

POP {R0-R7,PC}      ; restore link register to
ENDP               ; Programm Counter and return
```

4.4 uart_init

```
;*******************************************************************************
;*            U N T E R P R O G R A M M:    uart_init                          *
;*                                                                             *
;* Aufgabe:   Initialisiert USART2                                             *
;* Input:     R0....Baudrate                                                   *
;*******************************************************************************
init_uart       PROC
                push    {R0-R7,LR}      ;save link register to Stack
                LDR      R0,=4800       ;Baudrate

                LDR      R1,=RCC_APB2ENR          ; GPIOA mit einem Takt versorgen
                LDR      R1,[R1]
                ORR      R1,R1,#0x4
                LDR      R2,=RCC_APB2ENR
                STR      R1,[R2]

                LDR      R1,=GPIOA_CRL  ; loesche PA.2 (TXD-Leitung) configuration-bits
                LDR      R1,[R1]
                BIC      R1,R1,#0xF00
                LDR      R2,=GPIOA_CRL
                STR      R1,[R2]

                MOV      R1,R2          ; TX (PA2) - alt. out push-pull
                LDR      R1,=GPIOA_CRL
                LDR      R1,[R1]
                ORR      R1,R1,#0xB00
                STR      R1,[R2]

                MOV      R1,R2          ;loesche PA.3 (RXD-Leitung) configuration-bits
                LDR      R1,=GPIOA_CRL
                LDR      R1,[R1]
                BIC      R1,R1,#0xF000
                STR      R1,[R2]

                MOV      R1,R2              ;Rx (PA3) - inut floating
                LDR      R1,=GPIOA_CRL
                LDR      R1,[R1]
                ORR      r1,r1,#0x4000
                STR      R1,[R2]
```

```
        LDR     R1,=RCC_APB1ENR    ;USART1 mit einem Takt versorgen
        LDR     R1,[R1]
        ORR     R1,R1,#0x20000
        LDR     R2,=RCC_APB1ENR
        STR     R1,[R2]

        LDR     R1,=baudrate_const  ;Baudrate fuer USART festlegen
        LDR     R1,[R1]
        UDIV    R1,R1,R0
        LDR     R2,=USART2_BRR
        STRH    R1,[R2]

        LDR     R1,=USART2_CR1        ;aktiviere RX, TX
        LDR     R1,[R1]
        ORR     R1,R1,#0x0C           ;USART_CR1_RE = 1, (= Receiver Enable Bit)
        LDR     R2,=USART2_CR1        ;USART_CR1_TE = 1, (= Transmitter Enable Bit)
        STR     R1,[R2]

        LDR     R1,=USART2_CR1        ;aktiviere USART2
        LDR     R1,[R1]
        ORR     R1,R1,#0x2000         ;USART_CR1_UE = 1  (= UART Enable Bit)
        LDR     R2,=USART2_CR1
        STR     R1,[R2]
        pop     {R0-R7,PC}     ; save link register to Stack
        ENDP
```

4.5 alarm_wait

```
alarm_wait      PROC
                PUSH {R0-R2, LR}
                LDR R0, =0x63B      ;1 kHz
                LDR R1, =0x0
__alarm_loop    SUB R0, R0, #0x1
                CMP R0, R1
                BNE __alarm_loop
                POP {R0-R2, PC}
                ENDP
```

4.6 output_cnt

```
output_cnt      PROC
                PUSH{R0,LR}

                LDR R0, =GPIOB_ODR
                LSL R1, R9, #0x8
                STR R1, [R0]

                POP{R0,LR}

                ENDP
```

4.7 reset

```
reset           PROC
                PUSH{LR}

                LDR R8 , =0x0
                LDR R10, =0x0
                LDR R11, =0x0
                LDR R12, =0x0

                POP{PC}

                ENDP
```

4.8 inc_time

```
inc_time        PROC
                PUSH{LR}

                CMP R12, #0x0
                BEQ _no_inc

                ADD R11, R11, #0x1

_no_inc         POP{PC}

                ENDP
```

4.9 check_timeout

```
check_timeout   PROC
                PUSH{LR}

                LDR R8, =msg_time
                CMP R12, #0x0
                BEQ _not_timeout
                CMP R11, #0x64
                BEQ uart_put_string
                CMP R11, #0x64
                BNE _not_timeout
                BL reset

_not_timeout    POP{PC}

                ENDP
```

4.10 check_double

```
check_double    PROC
                PUSH{LR}

                CMP R10, #0x1
                BNE _not_dblfirst
                CMP R12, #0x1
                BNE _not_dblfirst
                LDR R8, =msg_dbbl
                BL uart_put_string
                BL reset

_not_dblfirst   CMP R10, #0x80
                BNE _not_double
                CMP R12, #0x80
                BNE _not_double
                LDR R8, =msg_dbbl
                BL uart_put_string
                BL reset

_not_double     POP{PC}

                ENDP
```

4.11 check_in

```
check_in        PROC
                PUSH{LR}

                CMP R10, #0x1
                BNE _not_first
                CMP R12, #0x0
                BNE _not_first
                LDR R12, =0x1
                LDR R8, =msg_gate
                BL uart_put_string



_not_first      CMP R10, #0x80
                BNE _no_checkin
                CMP R12, #0x0
                BNE _no_checkin
                LDR R12, =0x80
                LDR R8, =msg_gate
                BL uart_put_string

_no_checkin     POP{PC}

                ENDP
```

4.12 check_out

```
check_out       PROC
                PUSH{LR}

                CMP R10, #0x80              ; is second gate on
                BNE _not_second
                CMP R12, #0x01              ; is someone inside
                BNE _not_second
                LDR R12, =0x0               ; outside
                ADD R9, R9, #0x1            ; increment

                LDR R8, =msg_in
                BL uart_put_string
                BL output_cnt

_not_second     CMP R10, #0x01
                BNE _no_checkout
                CMP R12, #0x80
                BNE _no_checkout
                LDR R12, =0x0
                CMP R9 , #0x0               ;Check if overflow
                BEQ _no_checkout
                SUB R9, R9, #0x1

                LDR R8, =msg_out
                BL uart_put_string
                BL output_cnt

_no_checkout    POP{PC}

                ENDP
```

4.13 init_variables

```
;*******************************************************************************
;*            U N T E R P R O G R A M M:    init_variales                      *
;*                                                                             *
;* Aufgabe:   Initialisiert die Variablen counter, state, time und direction  *
;* Input:     keine                                                            *
;* return:    keine                                                            *
;*******************************************************************************

init_variables  PROC
                PUSH{LR}

                LDR R9, =0x0
                BL reset

                POP{PC}

                ENDP
```

4.14 wait_100ms

```
;*******************************************************************************
;*            U N T E R P R O G R A M M:    wait_100ms                         *
;*                                                                             *
;* Aufgabe:   Wartet 100ms                                                     *
;* Input:     keine                                                            *
;* return:    keine                                                            *
;*******************************************************************************

wait_100ms      PROC
                push    {R0-R2,LR}      ; save link register to Stack
                MOV     R0,#0x64        ; wait 100ms
                MOV     R1,#0
wait_ms_loop    MOV     R2,#0x63B
wait_ms_loop1   SUB     R2,R2,#1
                CMP     R2,R1
                BNE     wait_ms_loop1
                SUB     R0,R0,#1
                CMP     R0,R1
                BNE     wait_ms_loop
                POP     {R0-R2,PC}
               ;restore link register to Programm Counter and return

                ENDP
```

4.15 uart_put_char

```
;*******************************************************************************
;*            U N T E R P R O G R A M M:    uart_put_char                      *
;*                                                                             *
;* Aufgabe:   Ausgabe eines Zeichens auf USART1                                *
;* Input:     R0....Zeichen                                                    *
;* return:                                                                     *
;*******************************************************************************
uart_put_char      PROC
                   LDR      R1,=USART2_SR        ;Data Transmit Register leer?
                   LDR      R1,[R1]
                   TST      R1,#0x80     ; USART_SR_TXE = 1 (Last data already
                  BEQ       uart_put_char                    ; transferred ?)
                   LDR      R1,=USART2_DR
                   STR      R8,[R1]
                   BX       LR
                   ENDP
```

4.16 uart_put_string

```
;*******************************************************************************
;*            U N T E R P R O G R A M M:    uart_put_string                    *
;*                                                                             *
;* Aufgabe:   Ausgabe einer Zeichenkette (C Konvention) USART1                 *
;* Input:     R0....Zeiger auf Sting                                           *
;* return:                                                                     *
;*******************************************************************************
uart_put_string     PROC
                    PUSH     {LR}
                    MOV      R2,R8               ; R2 = Anfangsadresse von String
                    B        _check_eos
_next_char          LDRB     R8,[R2],#0x01
                    BL       uart_put_char
_check_eos          LDRB     R8,[R2,#0x00]
                    CMP      R8,#0x00          ; '\0' Zeichen ?
                    BNE      _next_char
                    POP      {PC}
                    ENDP

                    ALIGN

                    END
```

5 Testdaten

Schalterzustände	Wirkung	Anmerkung
1. Schranke dann 2. Schranke	Counter erhöht sich um 1	
2. Schranke dann 1. Schranke	Counter verringert sich um 1, außer Counter = 0	Musste nachträglich hinzugefügt werden
Beide Schranken gleichzeitig	Der Piezo fängt an zu läuten und Counter verändert sich nicht	
1. Schranke zweimal hintereinander aktiviert	Counter verändert sich nicht, Zählversuch wird verworfen	
2. Schranke zweimal hintereinander aktiviert	Counter verändert sich nicht, zählversuch wird verworfen	
1. Schranke Timeout (10s)	Counter verändert sich nicht, zählversuch wird verworfen	
2. Schranke Timeout (10s)	Counter verändert sich nicht, zählversuch wird verworfen	

6 Zeitaufwand

Maximilian Roll:

Tätigkeit	Aufwand
Erstellung des Pflichtenhefts	0,5h
Erstellung des Systemdesigns	1h
Programmcodierung	4
Testen der Software	8
Dokumentation	2
Gesamt:	**15,5**

Martin Platajs:

Tätigkeit	Aufwand
Erstellung des Pflichtenhefts	0,5h
Erstellung des Systemdesigns	1h
Programmcodierung	2
Testen der Software	6
Dokumentation	4
Gesamt:	**13,5**

7 Aufgetretene Probleme

Die V24-Schnittstelle hat anfänglich nichts ausgegeben. Die Warteschleife war mit 500ms zu lang und wurde auf 100ms reduziert. Wenn Counter 0 war es möglich rückwärts zu zählen.

8 Erkenntnisse aus der Übung

- Umgang mit der Hardware Cortex-M3
- Besseres Verständnis der Assemblerprogrammierung
- Gemeinsames Arbeiten an einem Projekt