



HÖHERE TECHNISCHE BUNDESLEHRANSTALT

HOLLABRUNN

Höhere Abteilung für Elektronik – Technische Informatik

Klasse / Jahrgang: 4BHEL	Übungsbetreuer: Mag. Dipl.-Ing. Wihsböck Michael
Übungsnummer: 4	Übungstitel: Inkrementalgeber
Datum der Vorführung: 10.06.2020	Gruppe: Maximilian Roll, Martin Platajs
Datum der Abgabe: 18.06.2020	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	Originalangabe	3
2	Pflichtenheft	4
2.1	Inkrementalgeber	4
2.2	Bitmuster	4
3	Blockschaltbild	5
4	Kommentierter Quellcode	6
4.1	Allgemeines	6
4.2	set_clock_32MHz	7
4.3	TIM4_IRQHandler	7
4.4	TIM3_IRQHandler	8
4.5	TIM4_Config	8
4.5.1	Timer 4 berechnen	9
4.6	TIM3_Config	9
4.6.1	Timer 3 berechnen	9
4.7	NVIC_init	10
4.8	init_ports	10
4.9	Clock	11
4.10	init_uart	12
4.11	changed_pos	12
4.12	main	13
5	Testplan	14
5.1	Taktfrequenz	14
5.2	Genauigkeit der Uhr	14
6	Zeitaufwand	15
7	Aufgetretene Probleme	15
8	Ergebnis	15

Übungstitel: **Inkrementalgeber**

Übungsdatum:

Hardwarekomponente: Inkrementalgeber ARM Minimalsystem

Uhr mit Timer: Timer 4

Zu verwendender Systemtakt: HSE 32 Mhz

Entwickle in der Programmiersprache C eine interrupt gesteuerte Software für den Cortex-M3 Mikrocontroller welche mehrere parallele Prozesse realisiert. (Echtzeitsystem)
 Folgende Struktur soll dabei realisiert werden

Prozess A: Hauptprg, Initialisierung, Visualisierung von Messwerten und Uhr auf Anzeige**Prozess B:** Uhrenfunktion (gemeinsamer Speicher mit Prozess A)**Prozess C:** Erfassung von Messdaten (z.B.: Drehzahlmessung)

Am Beginn des Programms soll zunächst eine **Begrüßungstext** ausgegeben werden, der zu einem Hardwaretest auffordert (Komponente vorhanden oder nicht vorhanden)

Allgemeine Regeln für diesen Übungsdurchgang:

- Es ist ein detailliertes Pflichtenheft zu erarbeiten
- Zuerst ist als Programmbeschreibung ein **Blockschaltbild** zu zeichnen, welches **alle** Information in kommentierte graphischer Form darstellt.
- Timer, Counter Konstanten sind zu berechnen und graphisch darzustellen
- Uhrzeit ist im ASCII Code in folgender Form anzuzeigen **hh:mm:ss:z**
- Vorsicht: print sind nicht reentrant !! und soll deshalb in keiner ISR verwendet werden
- Der Source Code ist entsprechend zu **dokumentieren** bzw. soll es für jede Funktion einen entsprechenden **Funktionskopf** gegen der Funktion beschreibt (Aufgabe der Funktion, Input bzw. Output Parameter und eventuelle Error Codes) – idealerweise nach doxygen Standard
- Sinnvoll ist für die Fehlersuche bzw. anschließend für den Funktionsnachweis Oszilloskop bzw. Logikanalysator eventuell einzusetzen

Angabebesprechung, Schriftliches, detailliertes Pflichtenheft (incl. Bitbelegung und Timing)	
Blockschaltbild	
Vorführung des lauffähigen Programms, Testdatensatz und Demonstration der Tabelle ist Bestandteil der Vorführung	
Protokollabgabe	

Protokollaufbau (Reihenfolge und Nummerierung einhalten!!!!):

- 1.) Inhaltsverzeichnis
- 2.) Originalangabe und Unterschriften
- 3.) Pflichtenheft (Angabekonkretisierung, -erweiterung, -einschränkung, -änderung),
Bitbelegung
- 4.) Algorithmusbeschreibung: Beschreibung des Gesamtsystems mithilfe eines
Blockschaltbildes , ev. mit Foto
- 5.) Kommentiertes Listing bzw. Header Datei der Library (*.h)
- 6.) Testplan (Nachweis der einzelnen Teilfunktionen, Wirkung extremer Eingaben, Grenzfälle,
wann kommt's zum Absturz ...)
- 7.) Zeitaufwand (aufgeschlüsselt) und Arbeitsteilung
- 8.) Aufgetretene Probleme
- 9.) Betrachtung der Ergebnisse und Erkenntnisse aus der Übung

2 Pflichtenheft

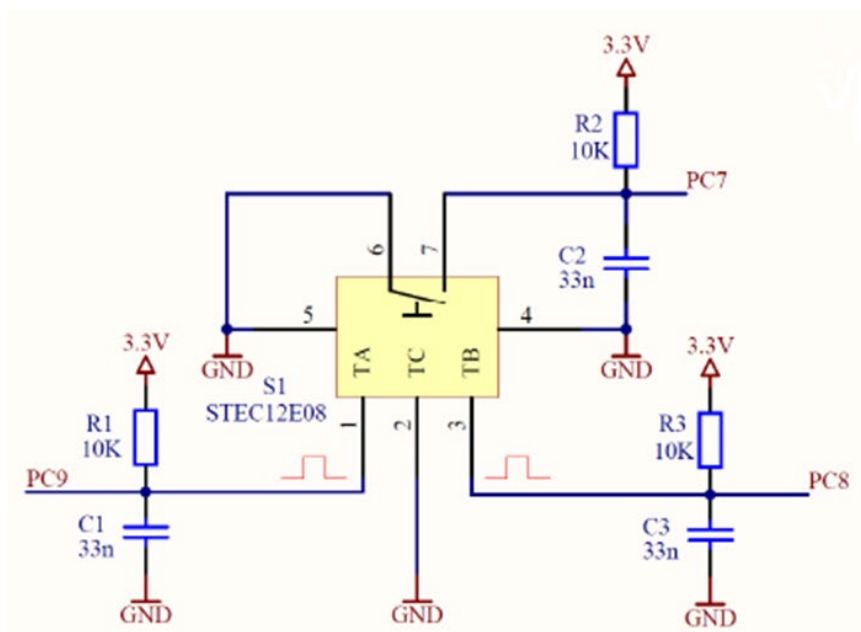
Es soll eine interrupt-gesteuerte Software geschrieben werden, welche auf einer Anzeige die Uhrzeit und die Daten der Drehzahlmessung anzeigt. Der anzusteuernende Inkrementalgeber (STEC12E08) ist bereits auf dem CM3 Basisboard verbaut. Zu Beginn soll ein Begrüßungstext erscheinen und die Funktion der Hardware getestet werden. Zusätzlich soll der SystemClock des Cortex auf 32Mhz hoch getaktet werden. Der Code soll in 3 Prozessen eingeteilt werden:

Prozess A: Hauptprogramm

Prozess B: Uhrenfunktion

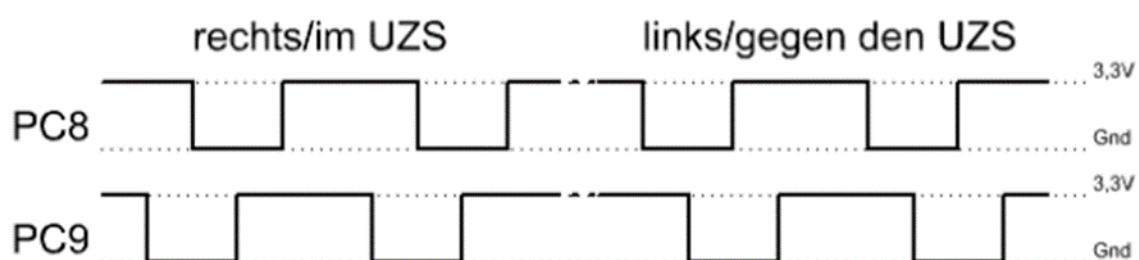
Prozess C: Erfassung von Messdaten

2.1 Inkrementalgeber

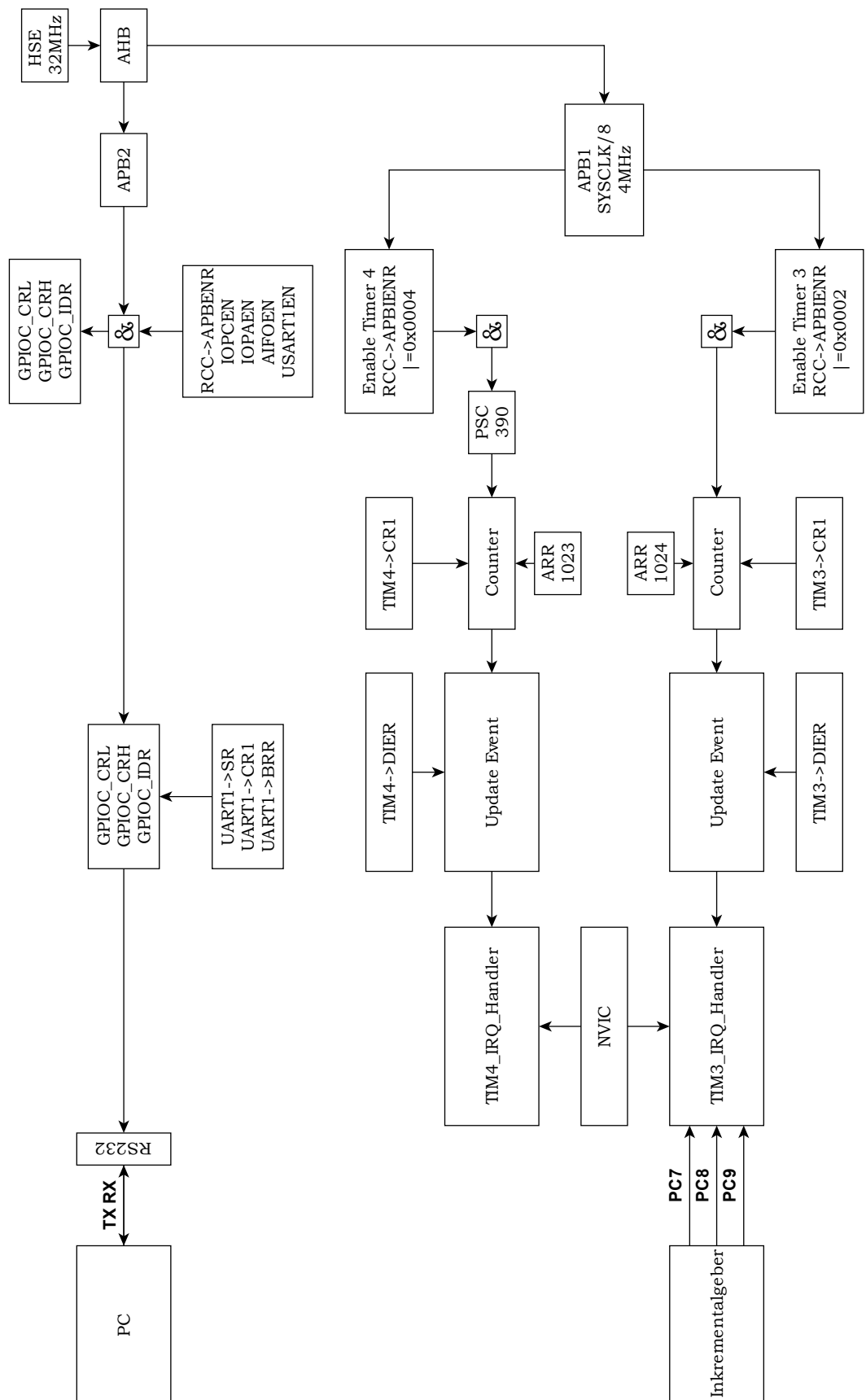


Der Inkrementalgeber ist an den Ports PC7, PC8 und PC9 angeschlossen. Der Button des Inkrementalgebers ist mit PC7 verbunden. Die Signale A und B liegen an den Ports PC8 und PC9 and und liefern folgendes Signalmuster:

2.2 Bitmuster



3 Blockschaltbild



4 Kommentierter Quellcode

4.1 Allgemeines

```

1  /*****
2  /*  (C) Copyright HTL - HOLLABRUNN  2020 All rights reserved. AUSTRIA      */
3  /*
4  /* File Name:  Inkrementalgeber.c
5  /* Autor:      Maximilian Roll / Martin Platajs
6  /* Version:    V1.04.4
7  /* Date:       04/05/2020
8  /* Description: displays clock and position of rotary encoder
9  *****/
10 /* History:
11 /*V1.00  creation
12 /*V1.01.0 working clock
13 /*V1.02.0 working rotary encoder
14 /*V1.03.0 other version of rotary encoder
15 /*V1.04.0 final working version of rotary encoder
16 /*V1.04.1 bug fixes and optimizations
17 /*V1.04.1 bug fixes and optimizations
18 /*V1.04.3 fixed clock prescaler and ARR
19 /*V1.04.4 comments and final optimizations
20 *****/
21 #include <armv10_std.h>
22
23 /*----- Function Prototypes -----*/
24 static void NVIC_init(char position, char priority);
25 static void TIM4_Config(void);
26 static void TIM3_Config(void);
27 static void set_clock_32MHz(void);
28 static void clock(void);
29 static void changed_pos(void);
30 static void init_ports(void);
31 void init_uart(unsigned long);
32
33 /*----- Bit Banding Alias Adresses -----*/
34 #define GPIOC_IDR  GPIOC_BASE + 2*sizeof(uint32_t)           // Calc peripheral address GPIOC IDR
35 #define BITBAND_PERI(a,b) ((PERIPH_BB_BASE + (a-PERIPH_BASE)*32 + (b*4)))
36 #define PC7  *((volatile unsigned long *) (BITBAND_PERI(GPIOC_IDR,7))) //Button of rotary encoder
37 #define PC8  *((volatile unsigned long *) (BITBAND_PERI(GPIOC_IDR,8))) //A-lane of rotary encoder
38 #define PC9  *((volatile unsigned long *) (BITBAND_PERI(GPIOC_IDR,9))) //B-lane of rotary encoder
39
40 /*----- Static Variables-----*/
41 volatile int position=0;           //position of rotary encoder
42 volatile int ackn = 2;             //acknowledge for rotary encoder ("2" == ready, "1" == in counter clockwise turn, "0" == in clockwise turn)
43 volatile int rpos=0;              //position in clockwise turn      ("10"->"0"   "00"->"1"   "01"->"2"   "11"->"3")
44 volatile int lpos=0;              //position in counter clockwise turn  ("01"->"0"   "00"->"1"   "10"->"2"   "11"->"3")
45 volatile int ds;                  //counter for clock
46

```

Hier sieht man alle Prototypen für die jeweiligen Unterprogramme, weiters werden die Ports PC7, PC8 und PC9 gemeinsam mit den globalen Variablen definiert. Außerdem wurde die Library armv10_std.h hier inkludiert.

4.2 set_clock_32MHz

```

47  /*****
48  /*          Set Systemclock to 32 MHz          */
49  /*****
50  static void set_clock_32MHz(void)      //overclock SystemClock to 32 MHz
51  {
52      RCC->CR |= RCC_CR_HSEON;           //HSE on
53      while ((RCC->CR & RCC_CR_HSERDY) == 0); //waiting for HSE being ready (HSERDY=1)
54
55      RCC->CFGR |= RCC_CFGR_PLLMULL4;     //multiply by 4 -> 4 times 8 MHz equals 32 MHz (SYSCLK)
56      RCC->CFGR |= RCC_CFGR_PPRE1_DIV16;  //PCLK1 (APB1)=SYSCLK/8 -> 4MHz
57      RCC->CFGR |= RCC_CFGR_PLLSRC;
58
59      RCC->CR |= RCC_CR_PLLON;           //PLL on
60      while ((RCC->CR & RCC_CR_PLLRDY) == 0); //wating for PLL being ready (PLLRDY=1)
61
62      RCC->CFGR |= RCC_CFGR_SW_PLL;       //PLL = Systemclock
63      while ((RCC->CFGR & RCC_CFGR_SWS_PLL) == 0);
64      /*      waiting till SYSCLK is stabalized      */
65
66      while ((RCC->CFGR & RCC_CFGR_SWS) != ((RCC->CFGR<<2) & RCC_CFGR_SWS));
67
68      RCC->BDCR |=RCC_BDCR_LSEON;        //32 kHz for RTC (AN2821 Reference Manual)
69  }

```

In diesem Unterprogramm wird zunächst der **H**igh **S**peed **E**xternal Oszillator kurz HSE aktiviert und dessen Signal auf 32 MHz hoch getaktet. Außerdem wird PCLK1 auf 4 MHz herabgesetzt. Dann wird der PLL initialisiert und sichergestellt, dass er stabil ist. Schließlich wird der **L**ow **S**peed **E**xternal Oszillator kurz LSE aktiviert, welcher die **R**eal **T**ime **C**lock, RTC, mit einem Taktsignal versorgt.

4.3 TIM4_IRQHandler

```

71  /*****
72  /*  Interrupt Service Routine  Timer4 (triggers every 0.1s and counts ds)  */
73  /*****
74  void TIM4_IRQHandler(void) //timer 4, triggers every 0.1s
75  {
76      TIM4->SR &=~0x01;    // delete interrupt pending bit (prevents another interrupt)
77      ds++;
78
79  }

```

Diese Interrupt Service Routine, ISR, wird durch den Timer 4 alle 100ms ausgelöst. Dabei wird das Pending-Bit zurückgesetzt und die Variable ds(decisecond) um 1 erhöht.

4.4 TIM3_IRQHandler

```

81  /*****
82  /* Interrupt Service Routine Timer3 (checks for rotary encoder movement) */
83  /*****
84  void TIM3_IRQHandler(void) //timer 3, refreshes and checks for status of rotary encoder
85  {
86      TIM3->SR &=~0x01; // delete interrupt pending bit (prevents another interrupt)
87
88      if(PC8 == 1 & PC9 == 0 & rpos == 0 & ackn == 2){rpos=1;ackn=0;} //set clockwise ackn(0), step 1 (10)
89      else if(PC8 == 0 & PC9 == 0 & rpos == 1 & ackn == 0){rpos=2;} //step 2 (00)
90      else if(PC8 == 0 & PC9 == 1 & rpos == 2 & ackn == 0){rpos=3;} //step 2 (01)
91      else if(PC8 == 1 & PC9 == 1 & rpos == 3 & ackn == 0){rpos=0;ackn=2;position--;} //set ready ackn(2), count position down, step 4 (11)
92
93      if(PC8 == 0 & PC9 == 1 & lpos == 0 & ackn == 2){lpos=1;ackn=1;} //set counter clockwise ackn(1), step 1 (01)
94      else if(PC8 == 0 & PC9 == 0 & lpos == 1 & ackn == 1){lpos=2;} //step 2 (00)
95      else if(PC8 == 1 & PC9 == 0 & lpos == 2 & ackn == 1){lpos=3;} //step 3 (10)
96      else if(PC8 == 1 & PC9 == 1 & lpos == 3 & ackn == 1){lpos=0;ackn=2;position++;} //set ready ackn(2), count position up, 4. step 4 (11)
97
98      if(PC7 == 0){position = 0;} //reset position if button pressed
99  }

```

Diese Interrupt Service Routine, ISR, wird durch den Timer 3 alle 256µs ausgelöst. Dabei wird der momentane Status des Inkrementalgebers überprüft. Es wird überprüft ob eine Drehung im oder gegen den Uhrzeigersinn eingeleitet wird und es wird geprüft ob der Button des Inkrementalgebers gedrückt wird. Sollte der Inkrementalgeber sich in einer Drehung befinden, wird mit dem Acknowledge überprüft ob es sich um eine Drehung im oder gegen den Uhrzeiger handelt und die Variablen rpos bzw. lpos geben an bei welchem Bit Muster der Bitfolge man sich gerade befindet. Am Ende einer Drehung wird das Acknowledge auf „READY“ gesetzt, die Position des Inkrementalgebers hoch oder herabgezählt und die Position (rpos bzw. lpos) in der Bitfolge auf 0 gesetzt.

4.5 TIM4_Config

```

101  /*****
102  /* Initialization Timer4 (triggers every 0.1s) */
103  /*****
104  static void TIM4_Config(void)
105  {
106      /*----- config timer 4 -----*/
107      RCC->APB1ENR |= 0x0004; //timer 4 clock enable
108      TIM4->SMCR = 0x0000; //timer 4 clock selection: CK_INT
109      TIM4->CR1 = 0x0000; //choose timer mode: upcounter
110
111      /*Timer 4 -> 4MHz (look 32MHz config)
112      Prescaler = 390 & auto reload value = 1023 equals in 0.1s refresh*/
113      TIM4->PSC = 390; //prescaler value
114      TIM4->ARR = 1023; //auto reload value
115      TIM4->RCR = 0; //deactivate repetition counter
116
117      /*----- config interrupt timer 4 -----*/
118      TIM4->DIER = 0x01; //enable interrupt on overflow/underflow
119      NVIC_init(TIM4_IRQn,1); //enable timer 4 update interrupt, priority 1
120
121      /*----- start Timer 4 -----*/
122      TIM4->CR1 |= 0x0001; //set counter-enable bit
123  }

```

In diesem Unterprogramm wird der Timer 4 konfiguriert damit dieser alle 100ms auslöst. Hierfür wird dieser aktiviert, mit dem internen Clock versorgt, und auf upcounter Modus gesetzt. Der Prescaler wird mit 390 und der Auto Reload Value auf 1023 definiert. Weiters wird eingestellt, dass nach einem Update Event nicht gestoppt wird, dass bei einem Überlauf des Counters der Interrupt ausgelöst wird und dass der Interrupt mit einer Priorität von 1 aktiviert wird. Schließlich wird der Counter aktiviert.

4.5.1 Timer 4 berechnen

$$T_{such} = 0,1$$

$$f_{sysclk} = 32 \text{ MHz}$$

$$PSC = 390$$

$$ARR = 1023$$

$$f_{Timer} = \frac{f_{sysclk}}{8} = 4 \text{ MHz} \rightarrow T_{Timer} = \frac{1}{f_{Timer}} = 250 \text{ ns}$$

$$T_{such} = T_{Timer} * (PSC + 1) * ARR = 0,09999825 \text{ s}$$

4.6 TIM3_Config

```

125  /******
126  /*          Initialization Timer3 (triggers every 256us)          */
127  /******
128  static void TIM3_Config(void)
129  {
130      /*----- config timer 3 -----*/
131      RCC->APB1ENR |= 0x0002; //timer 3 clock enable
132      TIM3->SMCR = 0x0000;    //timer 3 clock selection: CK_INT
133      TIM3->CR1 = 0x0000;     //choose timer mode: upcounter
134
135      /*Timer 3 -> 4MHz (look 32MHz config)
136      no prescaler & auto reload value = 1024 equals in 256us refresh*/
137      TIM3->ARR = 1024;       //auto reload value
138      TIM3->RCR = 0;          //deactivate repetition counter
139
140      /*----- config interrupt timer 3 -----*/
141      TIM3->DIER = 0x01;      //enable interrupt on overflow/underflow
142      NVIC_init(TIM3_IRQn,2); //enable timer 3 update interrupt, priority 1
143
144      /*----- start Timer 3 -----*/
145      TIM3->CR1 |= 0x0001;    //set counter-enable bit
146  }

```

In diesem Unterprogramm wird der Timer 3 konfiguriert damit dieser alle 256us auslöst. Hierfür wird dieser aktiviert, mit dem internen Clock versorgt, und auf upcounter Modus gesetzt. Der Prescaler wird nicht gesetzt und der Auto Reload Value auf 1024 definiert. Weiters wird eingestellt, dass nach einem Update Event nicht gestoppt wird, dass bei einem Überlauf des Counters der Interrupt ausgelöst wird und dass der Interrupt mit einer Priorität von 2 aktiviert wird. Schließlich wird der Counter aktiviert.

4.6.1 Timer 3 berechnen

$$T_{such} = ? \text{ (muss nur schnell genug sein)}$$

$$f_{sysclk} = 32 \text{ MHz}$$

$$PSC = 0$$

$$ARR = 1024$$

$$f_{Timer} = \frac{f_{sysclk}}{8} = 4 \text{ MHz} \rightarrow T_{Timer} = \frac{1}{f_{Timer}} = 250 \text{ ns}$$

$$T_{such} = T_{Timer} * (PSC + 1) * ARR = 256 \text{ us}$$

4.7 NVIC_init

```

148  /*****
149  /*          NVIC_init(char position, char priority)          */
150  /* Function:                                          */
151  /* completely initializes an interrupt in the nested      */
152  /* vectored interrupt controller (sets priority, prevents triggering, */
153  /* enables interrupt)                                  */
154  /* Übergabeparameter: "position" = 0-67 (interrupt number)      */
155  /* "priority": 0-15 (priority of the interrupt)                */
156  *****/
157  static void NVIC_init(char position, char priority)
158  {
159      NVIC->IP[position]=(priority<<4);                //Interrupt priority register: set interrupt priority
160      NVIC->ICPR[position >> 0x05] |= (0x01 << (position & 0x1F)); //Interrupt Clear Pending Register: prevents Interrupt from triggering when enabled
161      NVIC->ISER[position >> 0x05] |= (0x01 << (position & 0x1F)); //Interrupt Set Enable Register: Enable interrupt
162  }

```

In diesem Unterprogramm werden ISRs im NVIC initialisiert. Zunächst wird die Priorität gesetzt, dann wird verhindert, dass das Interrupt ausgelöst wird nachdem dieser aktiviert wurde. Schließlich wird der Interrupt aktiviert.

4.8 init_ports

```

167  static void init_ports(void)
168  {
169      int temp;
170
171      RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;           // enable clock for GPIOC (APB2 Peripheral clock enable register)
172
173      temp = GPIOC->CRL;
174      temp &= 0xFFFFFFFF; // delete PC7 config bit
175      temp |= 0x40000000; // define PC7 as Input floating
176      GPIOC->CRL = temp;
177
178      temp = GPIOC->CRH;
179      temp &= 0xFFFFF00; // delete PC8,PC9 config bits
180      temp |= 0x00000044; // define PC8,PC9 as input floating
181      GPIOC->CRH = temp;
182  }

```

Hier werden die Ports PC7, PC8 und PC9 die am Inkrementalgeber angeschlossen sind als Input floating initialisiert. Außerdem wird GPIOC mit einem Takt versorgt.

4.9 Clock

```

184  /*****
185  /*          clock(void)          */
186  /*  Function:          */
187  /*    formats and prints clock for lcd output          */
188  /*****
189  static void clock(void)
190  {
191      //variables
192      int lcd_ds;
193      int sec;
194      int lcd_sec;
195      int min;
196      int lcd_min;
197      int h;
198      int lcd_h;
199      char buffer[30];
200
201      //form clock
202      ds--;
203      lcd_ds=ds%10;
204      sec=ds/10;
205      lcd_sec=sec%60;
206      min=sec/60;
207      lcd_min=min%60;
208      h=min/60;
209      lcd_h=h%60;
210
211      //lcd output
212      lcd_set_cursor(0,0); // reset cursor
213      sprintf(&buffer[0],"%02d:%02d:%02d:%d", lcd_h, lcd_min, lcd_sec, lcd_ds); // refresh LCD
214      lcd_put_string(&buffer[0]);
215
216      if(lcd_h == 23 & lcd_min == 59 & lcd_sec == 59)
217      {
218          ds=0;
219      }
220  }

```

In diesem Unterprogramm wird die globale variable ds, die alle 0,1s um 1 erhöht wird, auf das Format hh:mm:ss:z umgerechnet und auf dem LCD Display ausgegeben.

4.10 init_uart

```

222  /*****
223  /*             initializes uart             */
224  /*****
225  void init_uart(unsigned long baudrate)
226  {
227      RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; //supply GPIOA with clock
228
229      GPIOA->CRH &= ~(0x00F0); //delete PA9 config
230      GPIOA->CRH |= (0x0BUL << 4); //define PA9-Tx as alternate function output push pull
231
232      GPIOA->CRH &= ~(0x0F00); //delete PA10 config
233      GPIOA->CRH |= (0x04UL << 8); //define PA10-Rx as input floating
234      RCC->APB2ENR |= RCC_APB2ENR_USART1EN; //supply USART1 with clock
235
236      USART1->BRR = 3200000L/baudrate; //baudrate needs now 3200000 because of the changed systemclock
237
238      USART1->CR1 |= (USART_CR1_RE | USART_CR1_TE); //activate RX, TX
239      USART1->CR1 |= USART_CR1_UE; //activate USART
240  }

```

Hier wird zunächst GPIOA mit einem Takt versorgt. Der Port PA9 fungiert als Tx-Leitung, somit wird dieser als alternate function output push pull definiert. Der Port PA10 hingegen ist die Rx-Leitung und wird deshalb als input floating definiert. Dann wird noch der USART1 mit einem Takt versorgt und Rx, Tx und USART werden aktiviert.

4.11 changed_pos

```

242  /*****
243  /*             updates uart and lcd if position has changed             */
244  /*****
245  static void changed_pos(void)
246  {
247      char buffer[30];
248      lcd_set_cursor(1,10);
249      lcd_put_string(" ");
250      lcd_set_cursor(1,10);
251
252      sprintf(&buffer[0], "%d", position); //update counter on lcd
253      lcd_put_string(&buffer[0]);
254      sprintf(&buffer[0], "Position: %d\r\n", position); //update counter on uart
255      uart_put_string(&buffer[0]);
256  }

```

In diesem Unterprogramm wird die aktuelle Position des Inkrementalgebers zum Display gebracht.

4.12 main

```

258  /******
259  /*                               MAIN function                               */
260  /******
261  int main (void)
262  {
263      int ds_test=0;
264      int position_test=0;
265
266      set_clock_32MHz();    // set systemclock to 32MHz
267
268      init_uart(9600);      // 9600,8,n,1
269      uart_clear();
270      USART1->CR1|=0x0020;  //USART1 RxNE - Interrupt Enable
271
272      lcd_init();           //init lcd display
273      lcd_clear();
274      ds=0;
275
276      init_ports();         //init Rotary Encoder Ports PC7, PC8 and PC9
277
278      uart_put_string("\r\nRotary Encoder V1.4.4-\r\n---press x to start---\r\n");
279      lcd_set_cursor(0,0);
280      lcd_put_string("00:00:00:0");
281      lcd_set_cursor(1,0);
282      lcd_put_string("Position: 0");
283
284      if(uart_get_char() == 'x' & PC8 == 1 & PC9 == 1)    //check Signal of Rotary Encoder and UART connection
285      {
286          TIM4_Config();    //init Timer 4 Config
287          TIM3_Config();    //init Timer 3 Config
288          uart_put_string("\r\nRotary Encoder connected and ready to use\r\n");
289          do
290          {
291              if (ds_test != ds)    //has ds changed? (every 0.1 sec)
292              {
293                  clock();          //update information
294                  ds_test=ds;       //reset test variable
295              }
296              if (position_test != position)    //has position changed? (if rotary encoder gets turned or pressed)
297              {
298                  changed_pos();    //update information
299                  position_test=position; //reset test variable
300              }
301          } while (1);             //endless loop
302      }
303  }

```

Im Hauptprogramm werden zunächst zwei Hilfsvariablen erstellt welche ein unnötiges Ausführen von Unterprogrammen verhindern soll. Dann werden nach und nach alle Konfigurationen aufgerufen und schließlich wird über UART eine Startsequenz eingeleitet mit welcher auch gleichzeitig überprüft wird ob vom Inkrementalgeber der normale High Pegel ankommt.

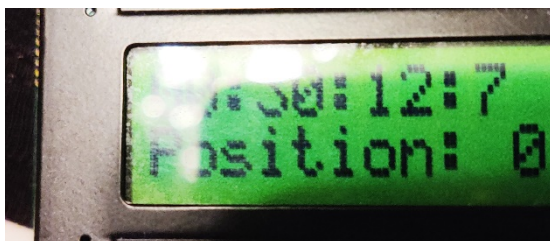
5 Testplan

5.1 Taktfrequenz

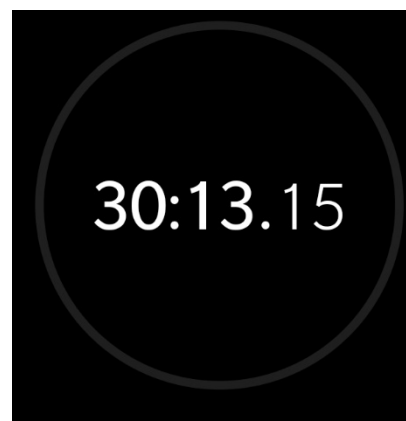
768 Byte Code Size Limit (18%)

Hier ist zu erkennen, dass der SYSClk auf den gewünschten 32 MHz taktet. Bei TIMXCLK wurde auf 4 MHz herunter getaktet da aus irgendeinem Grund bei einem höheren Takt die Uhr stehen bleibt, obwohl PSC und ARR richtig gesetzt waren.

5.2 Genauigkeit der Uhr



Mit einer Stoppuhr wurde die Genauigkeit gemessen. Die Uhr ist mehr oder weniger genau.



6 Zeitaufwand

Maximilian Roll:

Tätigkeit	Aufwand
Erstellung des Pflichtenhefts	0,5h
Erstellung des Systemdesigns	2h
Programmcodierung	8h
Testen der Software	5h
Dokumentation	5h
Gesamt:	20,5h

Martin Platajs:

Tätigkeit	Aufwand
Erstellung des Pflichtenhefts	0,5h
Erstellung des Systemdesigns	3h
Programmcodierung	12h
Testen der Software	5h
Dokumentation	3h
Gesamt:	24,5h

7 Aufgetretene Probleme

- Anfangs wurde die Messung des Inkrementalgebers über externe Interrupts gelöst, dann wurde versucht über den Encoder Modus des Timers die Position zu Messen. Nach kläglichem Scheitern wurde dann doch auf die einfache Variante gewechselt einen Timer immer wieder abfragen zu lassen ob sich beim Inkrementalgeber etwas tut.
- Die Zeitkonstanten PSC und ARR waren anfangs falsch berechnet
- Der UART wurde anfänglich falsch konfiguriert.

8 Ergebnis

Durch viel Ausprobieren und Einlesen vor allem beim Encode Mode bei Timern wurde sehr viel neues gelernt. Man ist jetzt noch vertrauter mit der CM3 Entwicklungsumgebung. Auch bei Externen Interrupts wurde viel Erfahrung gesammelt. Und das charakteristische Verhalten eines Inkrementalgebers wurde wieder in Erinnerung gerufen.