HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN

Höhere Abteilung für Elektronik – Technische Informatik

Klasse/ Jahrgang: 4AHEL	Übungsbetreuer: Mag. Dipl.-Ing. Wihsböck Michael
Übungsnummer: -	Übungstittel: Inkrementalgeber
Datum der Vorführung: 5.6.2019	Gruppe: Patrick Steiner, Matthias Breitseher
Datum der Abgabe: 6.6.2019	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: _____

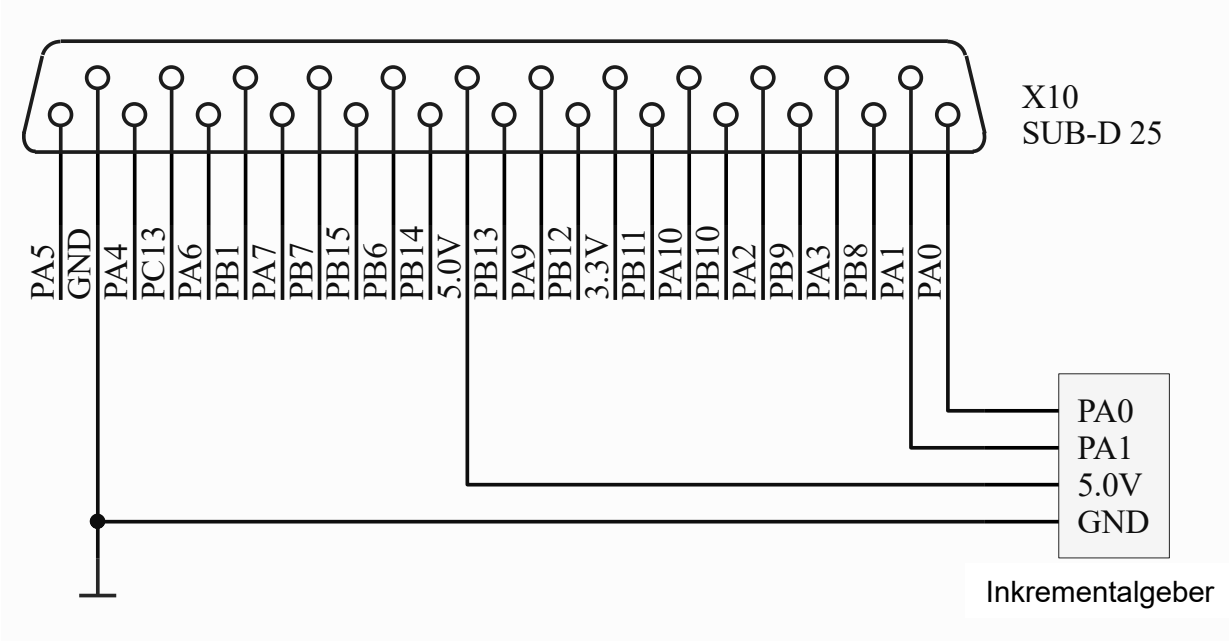
Inhaltsverzeichnis

1. Originalangabe

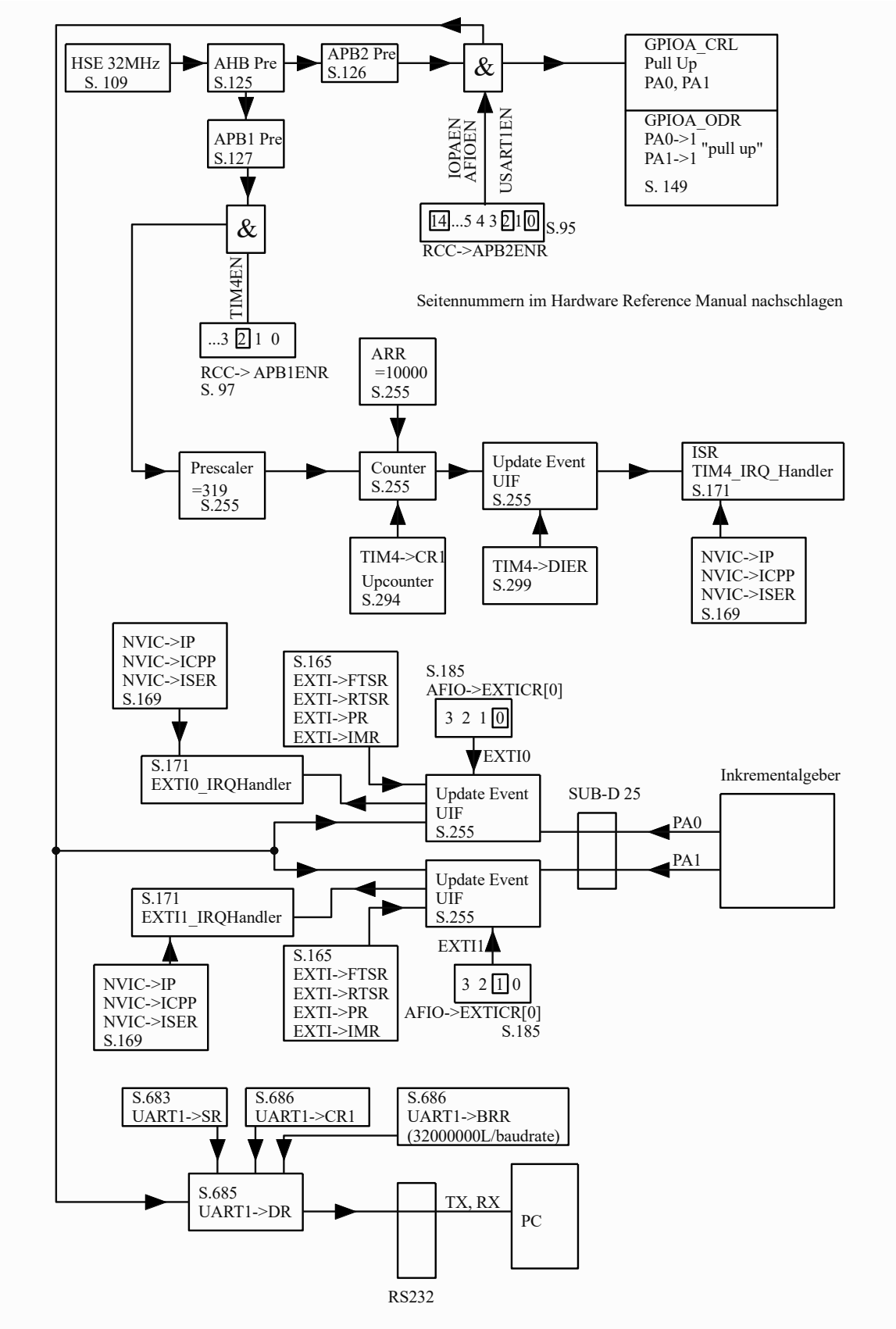
2. Pflichtenheft

Es soll eine interrupt gesteuerte Software geschrieben werden, welche auf einer Anzeige die Uhrzeit und die Daten der Drehzahlmessung anzeigt. Diese sollen in Interrupt Serviceroutinen realisiert werden und zu Beginn soll ein Begrüßungstext erscheinen. Zusätzlich soll der Timer4 für die Uhr verwendet werden und der Cortex auf 32Mhz rauf getaktet werden. Über den SUB-D25 Stecker kommuniziert der Drehzahlmesser, welcher aus einer Maus ausgebaut wurde, mit dem Cortex M3.

Bitbelegung



3. Blockschaltbild



3.1 Foto der Hardware

4. Algorithmusbeschreibung / Kommentiertes Listing

Zur besseren Übersicht wurden Algorithmusbeschreibung sowie die kommentierten Unterprogramme zusammengefasst:

```
/*****************************************************************************/
/*   (C) Copyright HTL - HOLLABRUNN  2009-2009 All rights reserved. AUSTRIA  */
/*                                                                           */
/* File Name:   Inkrementalgeber.c                                           */
/* Autor:     Matthias Breitseher / Patrick Steiner                          */
/* Version:   V5.00                                                          */
/* Date:    04/06/2019                                                       */
/* Description: Uhr und Positionsanzeige für Inkrementalgeber                */
/*****************************************************************************/
/* History:   V5.00  creation                                                */
/*****************************************************************************/
#include <armv10_std.h>

/*------------------------------ Function Prototypes -------------------------*/
static void NVIC_init(char position, char priority);
static void GPIOA_init(void);
static void TIM4_Config(void);
static void EXTI0_Config(void);
static void EXTI1_Config(void);
static void set_clock_32MHz(void);
static void Uhr(void);
static void position_up(void);
void uart_initialisierung(unsigned long);

/*------------------------------ Static Variables-----------------------------*/
volatile int links=0;
volatile int rechts=0;
volatile int ackr=0;
volatile int ackl=0;
volatile int zehntel;
volatile int position;
```

Dieser Programmteil zeigt die einzelnen Prototypen für die jeweiligen Unterprogramme, sowie die Festlegung der globalen Variablen und die Inkludierung der Library arm10, welche verwendet wurde um das Programm einfacher zu gestallten.

set_clock_32MHz

```
static void set_clock_32MHz(void)       //setzen den taktes auf 32MHz
{
  FLASH->ACR = 0x12;          // Flash latenxy setzen (2 waitstates)
  RCC->CR |= RCC_CR_HSEON;    //HSE on
  while ((RCC->CR & RCC_CR_HSERDY) == 0);
  //wait_10us for HSERDY=1 (HSE is ready)
  RCC->CFGR |= RCC_CFGR_PLLMULL4;   //4 mal 8 = 32 MHz (SYSCLK)
  RCC->CFGR |= RCC_CFGR_ADCPRE; //ADCCLK = SYSCLK* (APB2 PRESCALER=1)
  RCC->CFGR |= RCC_CFGR_PLLSRC;
  //  PLL= SYSCLK, HCLK=SYSCLK, da AHB PRESCALER=1
  RCC->CR |= RCC_CR_PLLON;    //PLL on
  while ((RCC->CR & RCC_CR_PLLRDY) == 0);
  //wait_10us for PLLRDY=1 (PLL is ready)
  RCC->CFGR |= RCC_CFGR_SW_PLL; //PLL = Systemclock
  while ((RCC->CFGR & RCC_CFGR_SWS_PLL) == 0);
  //wait till sysclk is stabilized (depending on selected clock)
  while ((RCC->CFGR & RCC_CFGR_SWS) != ((RCC->CFGR<<2) & RCC_CFGR_SWS));
  //end of stm32_ClockSetup
  RCC->BDCR |=RCC_BDCR_LSEON;
  //32kHz für RTC siehe AN2821 Referenze Manual 448ff/1072
}
```

Dieses Unterprogramm setzt den Takt des Cortex M3 von 8MHz zu den vorgegebenen 32MHz. Zunächst wird der High-Speed-External-Oszillator initialisiert. Anschließend wird überprüft ob HSE stabil ist (HSERDY=1). Als nächstes werden die Busse konfiguriert. PCLK2, welcher. APB2Peripherieeinheiten mit Takt versorgt, hat 32MHz. Schließlich muss die PLL noch konfiguriert werden (HSE*4) und der System Clock auf PLL gesetzt werden. Dann muss noch überprüft werden, ob der System Clock stabil ist (System Clock Switch Status = PLL). Als letzter Schritt noch im Backup Domain Controll Register (BDCR) der externe Low Speed Oscillator enabled (LSE). Dieser versorgt die Realtime Clock (RTC) mit einem Taktsignal.

TIM4_IRQHandler

```
56  /*******************************************************************************/
57  /*           Interrupt Service Routine  Timer4 (General Purpose Timer)         */
58  /*******************************************************************************/
59  void TIM4_IRQHandler(void)  //Timer 4, löst alle 100ms aus
60  {
61    TIM4->SR &=~0x01;
62    //Interrupt pending bit löschen
63    //(Verhinderung eines nochmaligen Interrupt-auslösens)
64    zehntel++;
65  }
66
```

Diese Interrupt Serviceroutine soll bei einem Timer Interrupt (welches alle 100ms geschieht) das Pending Bit löschen, um einen wieder Eintritt zu verhindern und die globale Variable zehntel um eins erhöhen.

EXTI1_IRQHandler

```
67  /*******************************************************************************/
68  /*           External Interrupt Service Routine  PA1                           */
69  /*******************************************************************************/
70  void EXTI1_IRQHandler(void)//ISR
71  {
72    EXTI->PR |= (0x01 << 1);
73    //Pending bit EXT0 rücksetzen (Sonst wird die ISR immer wiederholt)
74    if(rechts==1) //wenn zuvor PA0 eine nagative flanke hatte
75    {
76      ackr=1;   // rechtsdrehung
77      return;
78    }
79    else
80    {
81      links=1;
82      return;
83    }
84  }
```

Diese Interrupt Serviceroutine liegt an der Leitung PA1 und soll bei einem Interrupt zunächst das Pending Bit löschen, um einen wieder Eintritt zu verhindern. Die Hardware liefert Impulse, welche bei einer Positionsänderung auftreten, und auf den beiden Leitungen verzögert auftreten, um so die Richtung zu bestimmen. Zuerst PA0, dann PA1 -> Rechtsdrehung (position--). Zuerst PA1, dann PA0 -> Linksdrehung (position++). Wenn zuvor EXRI0 aktiviert wurde, ist die Variable rechts =1 und diese ISR setzt ackr auf 1. Wird EXTI1 zuerst ausgelöst, ist die Variable links=1.

EXTI0_IRQHandler

```
85  /*******************************************************************************/
86  /*          External Interrupt Service Routine  PA0                            */
87  /*******************************************************************************/
88  void EXTI0_IRQHandler(void)//ISR
89  {
90    EXTI->PR |= (0x01 << 0);
91    //Pending bit EXT0 rücksetzen (Sonst wird die ISR immer wiederholt)
92    if(links==1)  //wenn zuvor PA1 eine negative flanke hatte
93    {
94      ackl=1;   //linksdrehung
95      return;
96    }
97    else
98    {
99      rechts=1;
100     return;
101   }
102 }
```

Diese Interrupt Serviceroutine liegt an der Leitung PA0 und soll bei einem Interrupt zunächst das Pending Bit löschen, um einen wieder Eintritt zu verhindern. Die Hardware liefert Impulse, welche bei einer Positionsänderung auftreten, und auf den beiden Leitungen verzögert auftreten, um so die Richtung zu bestimmen. Zuerst PA0, dann PA1 -> Rechtsdrehung (position--). Zuerst PA1, dann PA0 -> Linksdrehung (position++). Wenn zuvor EXRI1 aktiviert wurde, ist die Variable links =1 und diese ISR setzt ackl auf 1. Wird EXTI0 zuerst ausgelöst, ist die Variable rechts=1.

TIM4_Config

```
101 /*******************************************************************************/
102 /*                      Initialization Timer4 (General Purpose Timer)          */
103 /*******************************************************************************/
104 static void TIM4_Config(void)
105 {
106   /*---------------------- Timer 4 konfigurieren -----------------------*/
107   RCC->APB1ENR |= 0x0004; //Timer 4 Clock enable
108   TIM4->SMCR = 0x0000;  //Timer 4 Clock Selection: CK_INT wird verwendet
109   TIM4->CR1  = 0x0000;
110   // Auswahl des Timer Modus: Upcounter --> Zählt:
111   //von 0 bis zum Wert des Autoreload-Registers
112   /* T_INT = 126,26ns, Annahme: Presc = 150 --->
113   Auto Reload Wert = 52801 (=0xCE41) --> 1s Update Event*/
114   TIM4->PSC = 319;    //Wert des prescalers (Taktverminderung)
115   TIM4->ARR = 10000;    //Auto-Reload Wert =
116 //  Maximaler Zaehlerstand des Upcounters
117   TIM4->RCR = 0;      //Repetition Counter deaktivieren
118   /*--------------------- Interrupt Timer 4 konfigurieren ---------------------*/
119   TIM4->DIER = 0x01;     //enable Interrupt bei einem UEV (Überlauf / Unterlauf)
120   NVIC_init(TIM4_IRQn,1);    //Enable Timer 4 Update Interrupt, Priority 1
121   /*--------------------------- Timer 4 Starten -----------------------------*/
122     TIM4->CR1 |= 0x0001;   //Counter-Enable bit setzen
123 }
```

Dieses Unterprogramm konfiguriert den Timer4 des Cortex M3 so, dass die ISR TIM4_IRQHandler alle 100ms auslöst und für die Uhr Funktion genutzt werden kann. Zunächst wird der Takt aktiviert. Das SMCR Register wird mit 0x0 beschreiben, das heißt: Slave mode ist disabled, internal Trigger 0 wird verwendet, kein externer Trigger Filter wird verwendet und kein external Trigger prescaler. Das CR1 Register wird mit 0x0 beschreiben, dadurch wird ausgewählt: Edge-aligned-mode, upcounter, Counter

stoppt nicht bei update Event, Counter disabled. Der Prescaler wird auf den Wert 319 gesetzt und das Auto-Reload-Register auf 10000. Im DIER Register wird das Update interrupt Bit enabled. Mit dem Unterprogramm NVIC_init wird die ISR enabled mit der Priorität 1 und zum Schluss wird im Controll Register der Counter enabled.

Berechnung der Zeitkonstanten

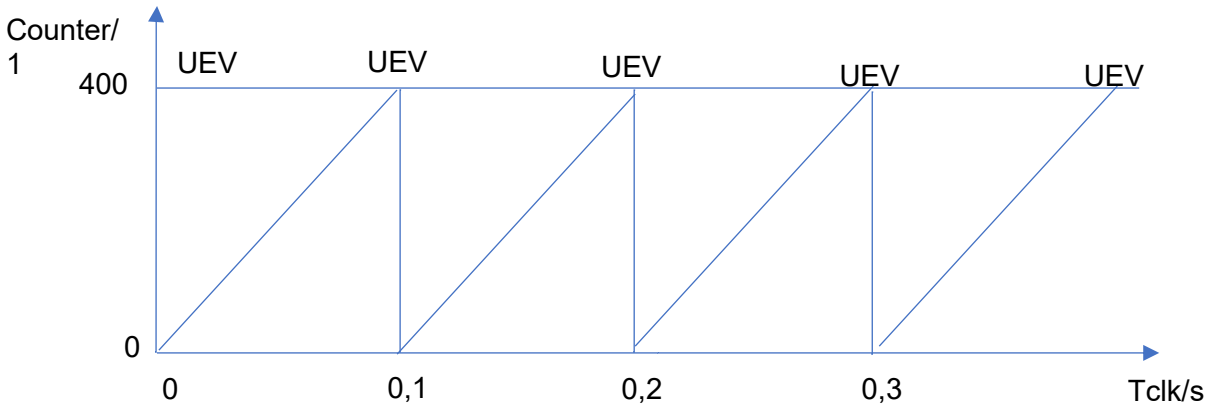
Tuev=0,1s
fsys=32MHz Tysy=1/fsys Tysy=3,125*10^(-8)

Tclk=Tsys *(PSC+1)
Tuev = Tsys*(PSC+1)*ARR

-> (PSC+1)*ARR = Tuev/Tsys (PSC+1)*ARR=0,1/3,125*10^(-8) =**3,2*10^(8)**

Gewählt:
PSC=**319**
ARR=**10000**

Grafik



EXTI1_Config

```
/*******************************************************************************/
/*                                  EXTI1_config                               */
/* Leitung PA1 wird mit EXTI1 verbunden, Interupt bei falling edge,Priorität 3*/
/*******************************************************************************/
static void EXTI1_Config()
{
  NVIC_init(EXTI1_IRQn,3);
  //NVIC fuer initialisieren EXTI Linel (Position 7, Priority 3)

  RCC->APB2ENR |= 0x0001;      //AFIOEN  - Clock enable
  AFIO->EXTICR[0] &= 0xFFFFFF0F;
  //Interrupt-Line EXTI1 mit Portpin PA1 verbinden
  EXTI->FTSR |= (0x01 << 1);     //Falling Edge Trigger für EXIT1 Aktivieren
  EXTI->RTSR &=~(0x01 << 1);     //Rising Edge Trigger für EXTI1 Deaktivieren

  EXTI->PR |= (0x01 << 1);
  //EXTI_clear_pending: Das Auslösen auf vergangene
//  Vorgänge nach dem enablen verhindern
  EXTI->IMR |= (0x01 << 1);
// Enable Interrupt EXTI1-Line. Kann durch den NVIC jedoch noch maskiert werden
}
```

Dieses Unterprogramm konfiguriert den externen interrupt für die Portleitung PA1. Zunächst wird mit dem UP NVIC_init die ISR initialisiert. Dann wird der interrupt mit Takt versorgt und die interrupt-Line EXTI1 wird mit Portpin PA1 verbunden. Anschließend wird auf falling edge getriggert und das Pending Bit wird gelöscht um einen wieder eintreten zu verhindern. Zum Schluss wird der interrupt enabled, er kann jedoch noch durch den NVIC maskiert werden.

EXTI0_Config

```
146  /*****************************************************************************/
147  /*                                  EXTI0_config                             */
148  /* Leitung PA0 wird mit EXTI0 verbunden, Interupt bei falling edge,Priorität 3*/
149  /*****************************************************************************/
150  static void EXTI0_Config()
151  {
152    NVIC_init(EXTI0_IRQn,3);
153    //NVIC fuer initialisieren EXTI Linel (Position 7, Priority 3)
154
155    RCC->APB2ENR |= 0x0001;      //AFIOEN  - Clock enable
156    AFIO->EXTICR[0] &= 0xFFFFFFF0;
157    //Interrupt-Line EXTIl mit Portpin PA0 verbinden
158    EXTI->FTSR |= (0x01 << 0);     //Falling Edge Trigger für EXITl Aktivieren
159    EXTI->RTSR &=~(0x01 << 0);     //Rising Edge Trigger für EXTIl Deaktivieren
160
161    EXTI->PR |= (0x01 << 0);
162    //EXTI_clear_pending: Das Auslösen auf
163  //  vergangene Vorgänge nach  dem enablen verhindern
164    EXTI->IMR |= (0x01 << 0);
165    // Enable Interrupt EXTIl-Line.
166  //  Kann durch den NVIC jedoch noch maskiert werden
167  }
```

Dieses Unterprogramm konfiguriert den externen interrupt für die Portleitung PA0. Zunächst wird mit dem UP NVIC_init die ISR initialisiert. Dann wird der interrupt mit Takt versorgt und die interrupt-Line EXTI1 wird mit Portpin PA0 verbunden. Anschließend wird auf falling edge getriggert und das Pending Bit wird gelöscht um einen wieder eintreten zu verhindern. Zum Schluss wird der interrupt enabled, er kann jedoch noch durch den NVIC maskiert werden.

NVIC_init

```
168  /*****************************************************************************/
169  /*                    NVIC_init(char position, char priority)                */
170  /* Funktion:                                                                 */
171  /*   Übernimmt die vollständige initialisierung eines Interrupts  im Nested  */
172  /*   vectored Interrupt controller (Priorität setzen, Auslösen verhindern,   */
173  /*   Interrupt enablen)                                                      */
174  /* Übergabeparameter: "position" = 0-67 (Nummer des Interrupts)              */
175  /*                    "priority": 0-15 (Priorität des Interrupts)            */
176  /*****************************************************************************/
177  static void NVIC_init(char position, char priority)
178  {
179    NVIC->IP[position]=(priority<<4);
180    //Interrupt priority register: Setzen der Interrupt Priorität
181    NVIC->ICPR[position >> 0x05] |= (0x01 << (position & 0x1F));
182    //Interrupt Clear Pendig Register: Verhindert,
183    //dass der Interrupt auslöst sobald er enabled wird
184    NVIC->ISER[position >> 0x05] |= (0x01 << (position & 0x1F));
185    //Interrupt Set Enable Register: Enable interrupt
186  }
```

Dieses UP initialisiert die ISR im NVIC. Zuerst wird die Priorität des Interrupts festgelegt. Um die entsprechende position für das Pending Bit und das Enable Bit zu erhalten, wird das richtige Register ausgewählt indem die position durch 32 dividiert wird (position 5x nach rechts verschieben) und anschließend das richtige Bit ausgewählt, indem die position mit Modulo 32 gerechnet wird (&0x1F).

GPIOA_init

```
187  /*****************************************************************************/
188  /*                                  GPIOA_init                               */
189  /* Leitung PA0 und PA1 werden als input definiert                            */
190  /*****************************************************************************/
191  static void GPIOA_init(void)
192  {
193    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
194    // enable clock for GPIOA (APB2 Peripheral clock enable register)
195    GPIOA->CRL &= 0xFFFFFF00;
196    // set Port Pins PA1 to Pull Up/Down Input mode (50MHz) = Mode 8
197    GPIOA->CRL |= 0x00000088;
198    GPIOA->ODR |= 0x0002;
199    //  Output Register für PA1 auf "1" initialisieren
200    GPIOA->ODR |= 0x0001;
201  }
```

In diesem UP werden die Leitungen PA0 und PA1 als Input definiert. Zunächst wird die GPIOA mit Takt versorgt. Anschließend werden die Bits für PA0 und PA1 gelöscht und als Input Pull Up definiert. Diese werden vom ODR auf 1 gesetzt, da sie pull up sind.

Uhr

```
202  /*****************************************************************************/
203  /*                                  Uhr                                      */
204  /* ausgabe und berechnung der aktuellen Zeit auf lcd display                 */
205  /*****************************************************************************/
206  static void Uhr(void)
207  {
208    int lcd_zehntel;    // aktueller Zählerstand auf LCD
209    int sekunden;       //berechnung der sekunden
210    int lcd_sekunden;   //variable zur ausgabe der sekunden
211    int minuten;//berechnung der minuten
212    int lcd_minuten;//variable zur ausgabe der minuten
213    int stunden;//berechnung der stunden
214    int lcd_stunden;//variable zur ausgabe der stunden
215    char buffer[30];
216
217    lcd_zehntel=zehntel%10;
218    sekunden=zehntel/10;
219    lcd_sekunden=sekunden%60;
220    minuten=sekunden/60;
221    lcd_minuten=minuten%60;
222    stunden=minuten/60;
223    lcd_stunden=stunden%60;
224
225    lcd_set_cursor(0,0);                // Cursor auf Ursprung
226    sprintf(&buffer[0],"%02d:%02d:%02d:%d", lcd_stunden, lcd_minuten, lcd_sekunden, lcd_zehntel);
227    // zehntelzaehler auf LCD aktualisieren
228    lcd_put_string(&buffer[0]);
229
```

```
230     if(lcd_stunden == 23 & lcd_minuten == 59 & lcd_sekunden == 59 & lcd_zehntel == 9)
231     {
232       zehntel=0;  // wenn 24h vergangen sind wird die zeit resetet
233     }
234   }
```

Dieses UP errechnet aus der variable zehntel, welches vom TIM4 alle 0,1 Sekunden erhöht wird, die Uhrzeit und gibt diese in hh:mm:ss:z auf der LCD Anzeige aus. Zunächst werden die benötigten Variablen zur Berechnung und Ausgabe der Variablen definiert sowie ein 30 Zeichen langes Array für die Ausgabe. Das Modulo 10 der variable zehntel ergibt die aktuelle zehntel Sekunde (in lcd_zehntel gespeichert). Modulo 60 von zehntel durch 10 ergibt die aktuellen Sekunden (in lcd_sekunden gespeichert). Die Minuten sind Sekunden /60 und das Ergebnis Modulo 60. Die Stunden sind Minuten/60 und das Ergebnis Modulo 60. Anschließend wird mit dem Cursor auf die Position 0,0 gegangen, die einzelnen Ergebnisse in den String Buffer zusammengeführt und ausgegeben. Zum Schluss wird kontrolliert ob 24h vergangen sind und wenn dem so ist, wird die Variable resettet.

Position_up

```
235   /*****************************************************************************/
236   /*                                  position_up                              */
237   /* ausgabe  aktuellen position auf lcd display                               */
238   /*****************************************************************************/
239   static void position_up(void)
240   {
241     char buffer[30];//hilfsvariable zur ausgabe des integers auf das lcd display
242     lcd_set_cursor(1,10);
243     lcd_put_string("       ");
244     lcd_set_cursor(1,10); //richtige cursor position setzen
245
246     sprintf(&buffer[0],"%d", position);//integer variable auf string schreiben
247     lcd_put_string(&buffer[0]);   //ausgabe des strings auf display
248     sprintf(&buffer[0],"\r\nPosition: %d", position);
249     uart_put_string(&buffer[0]);  //ausgabe der position per uart
250   }
```

Dieses UP gibt die aktuelle Position des Inkrementalgebers auf den LCD Display aus. Der Hilfsstring Buffer wird erstellt und mit dem Cursor auf die entsprechende Position gegangen. Der vorherige Text wird gelöscht und die Variable wird in den Buffer geschrieben. Diese wird ausgegeben, die Variable wird wieder in den Buffer geschrieben und per uart ausgegeben.

Uart_initialisierung

```
251 /*****************************************************************************/
252 /*                                uart_initialisierung                       */
253 /* initialisierung des uarts bei 32MHz                                       */
254 /*****************************************************************************/
255 void uart_initialisierung(unsigned long baudrate)
256 {
257   RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; //GPIOA mit einem Takt versorgen
258
259   GPIOA->CRH &= ~(0x00F0); //loesche PA.9 configuration-bits
260   GPIOA->CRH   |= (0x0BUL  << 4); //Tx (PA9) - alt. out push-pull
261
262   GPIOA->CRH &= ~(0x0F00); //loesche PA.10 configuration-bits
263   GPIOA->CRH   |= (0x04UL  << 8); //Rx (PA10) - floating
264   RCC->APB2ENR |= RCC_APB2ENR_USART1EN; //USART1 mit einem Takt versrogen
265
266   USART1->BRR  = 32000000L/baudrate;
267   //Baudrate festlegen statt 8000000 wie normal zu 32000000 ändern
268
269   USART1->CR1 |= (USART_CR1_RE | USART_CR1_TE);  //aktiviere RX, TX
270   USART1->CR1 |= USART_CR1_UE;                   //aktiviere USART
271
272 }
```

Da der interne Takt 32MHz und nicht 8MHz beträgt kann auf die uart Initialisierung in amv10 nicht zurückgegriffen werden. Zunächst wird die GPIOA mit Takt versorgt. Anschließend wird die Konfiguration für PA9 gelöscht und zu out push-pull gesetzt. Danach wird die Konfiguration für PA10 gelöscht und zu floating gesetzt. Darauffolgend, wird USART1 mit Takt versorgt und die Baudrate bei 32MHz ins BRR Register geschrieben. Zum Schluss wird Rx, Tx und USART aktiviert.

Main

```
273 /*****************************************************************************/
274 /*                                MAIN function                              */
275 /*****************************************************************************/
276 int main (void)
277 {
278   int zehntel_pruf=0;
279   int position_pruf=0;
280
281   set_clock_32MHz();
282   GPIOA_init();
283   uart_initialisierung(9600);    // 9600,8,n,1
284   USART1->CR1|=0x0020;      //USART1 RxNE - Interrupt Enable
285
286   lcd_init();         // LCD initialisieren
287   lcd_clear();        // Lösche LCD Anzeige
288   zehntel=0;    // zehntelzaehler initialisieren
289   TIM4_Config();
290   // Timer 1 starten: Upcounter --> löst alle 1s einen Update Interrupt
291   EXTI1_Config();
292   // Konfigurieren des Externen Interrupts für Leitung PA1 (Falling Edges)
293   EXTI0_Config();
294   // Konfigurieren des Externen Interrupts für Leitung PA0 (Falling Edges)
295   uart_put_string("Willkommen zum Inkrementalgeber:");  //begrüßungstext
296   lcd_set_cursor(1,0);
297   lcd_put_string("Position: 0");  //start text ausgeben
298   do
299   {
300     if (zehntel_pruf != zehntel)    // Hat sich Zählerstand verändert?
301     {
302       Uhr();
303       zehntel_pruf=zehntel;
304     }
```

```
305     if (position_pruf != position)  //hatt sich die position verändert?
306     {
307     position_up();
308     position_pruf=position;
309     }
310     if((links==1)&&(ackl==1))
311     {
312       position++;     //bei linksdrehung position --
313
314       links=0;
315       rechts=0;     //alle hilfsvariablen zurücksetzen
316       ackl=0;
317       ackr=0;
318
319     }
320     else if((rechts==1)&&(ackr==1))
321     {
322       position--;   //bei rechtsdrehung position ++
323
324       links=0;
325       rechts=0;   //alle hilfsvariablen zurücksetzen
326       ackl=0;
327       ackr=0;
328     }
329     } while (1);
330 }
331
```

Im Hauptprogramm werden zunächst die Variablen zehntel_pruf und position_pruf erstellt. Anschließend werden die vorher behandelten Unterprogramme für die entsprechenden Initialisierungen und Konfigurationen aufgerufen und das Uart1 RxNE Bit wird gesetzt für interrupt Enable. Der Willkommen Text wird per uart ausgegeben und der Starttext am LCD Display. Danach beginnt die Endlosschleife. Wenn es eine Änderung in den Variablen zehntel und position gibt, soll das entsprechende UP aufgerufen werden und die entsprechenden Hilfsvariablen wieder gleichgesetzt werden. Sind die Bedingungen für eine Linksdrehung erfüllt, also die variable links und ackl sind 1, wird die position erhöht und die Hilfsvariablen werden zurückgesetzt. Falls die Bedingungen für eine Rechtsdrehung erfüllt sind, rechts und ackr sind 1, wird die position verringert und die entsprechenden Variablen werden zurückgesetzt.

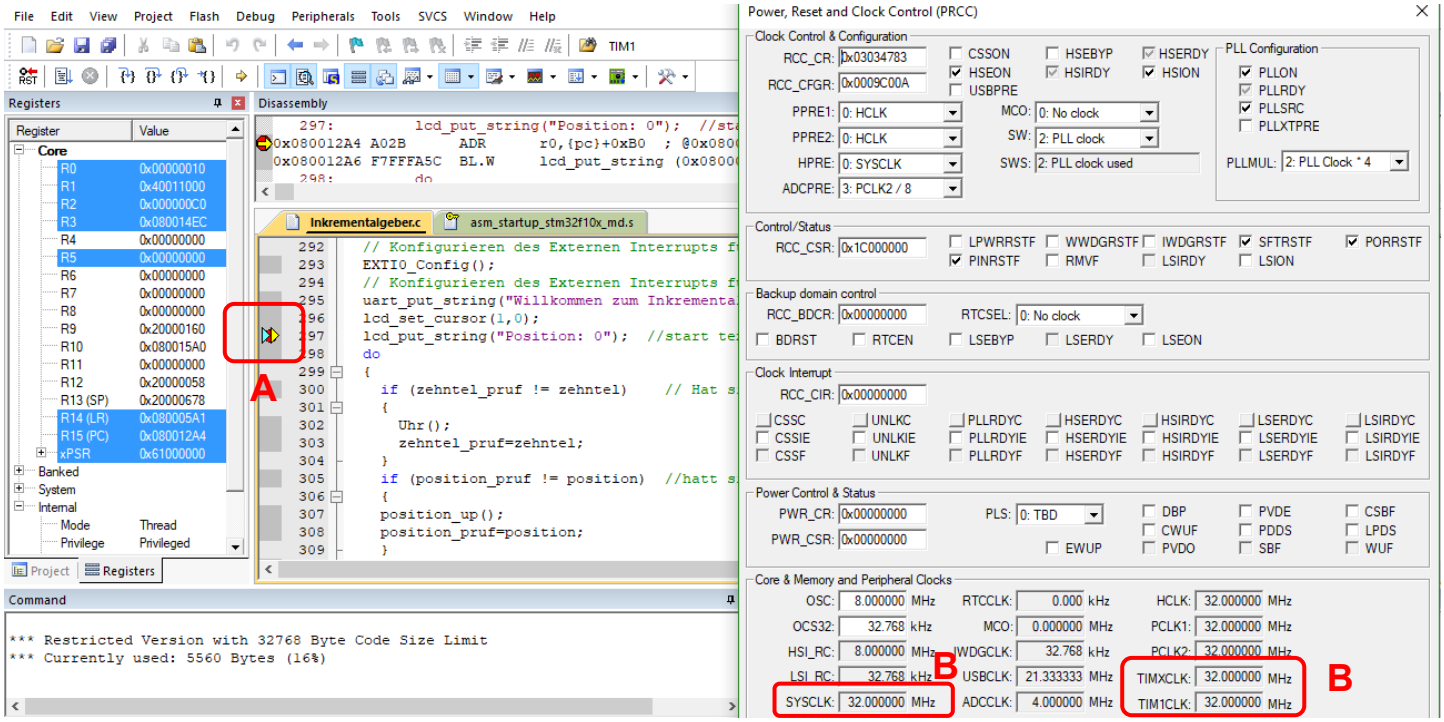
5.Testplan

Folgende Testfälle werden verwendet, um die Funktionalität des Programmes zu zeigen.

1. Nachweis Takt und Timer

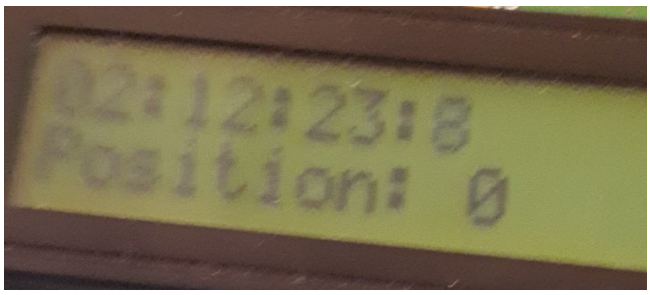
Durch setzen eines Breakpoints vor der Endlosschleife (**A**) im Debug Mode der uVision Arbeitsumgebung und anschließendem laufen des Programmes bis zu diesem Punkt mit Debug-> Run, kann mit Peripherals-> Power, Reset and Clock Control (PRCC) die Taktfrequenz nachgeschaut werden (**B**).

Das der Timer4 benutzt wurde und funktioniert kann man im Programm erkennen, da die TIM4_IRQHandler verwendet wurde und die Uhr funktioniert.



2. Nachweis der Uhr

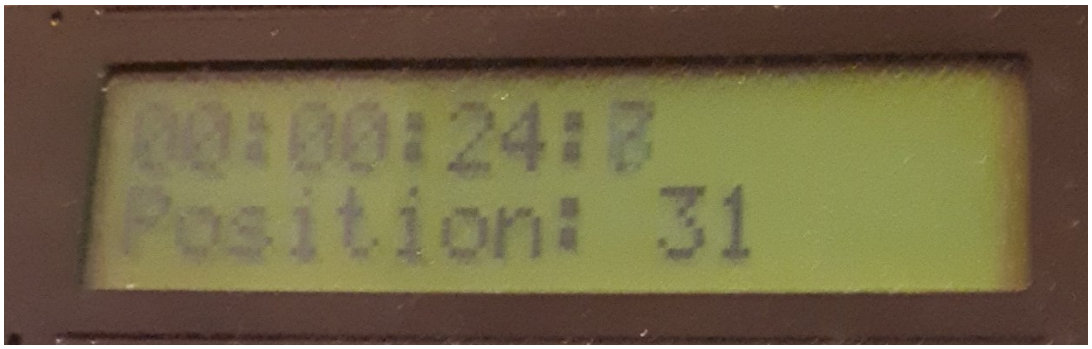
Um die Funktion zu testen wurde synchron zum Start des Programmes eine Handy Uhr gestartet und 2h 12min und 23 Sekunden lang laufen gelassen.



Wies zu sehen ist, ist die Abweichung minimal.

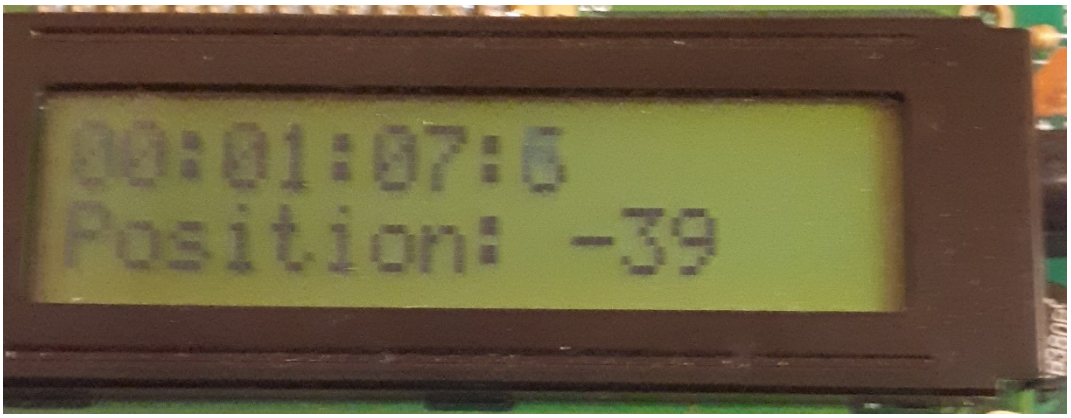
3. Nachweis Positionierung

Um zu beweisen, dass die Software hoch zählt, wenn der Inkrementalgeber nach links gedreht wird, wird der Inkrementalgeber nach links gedreht.



Wie zu sehen ist, erhöht sich die Position von der Ursprungs Position 0, wenn nach links gedreht wird.

Um zu beweisen, dass die Software runter zählt, wenn der Inkrementalgeber nach rechts gedreht wird, wird der Inkrementalgeber nach rechts gedreht.



Wie zu sehen ist, verringert sich die Position, wenn nach rechts gedreht wird.

4. Nachweis des Begrüßungstextes

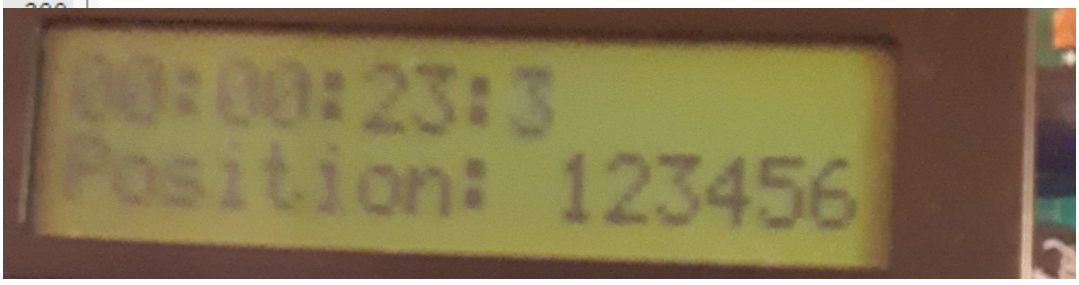
Um dies zu beweisen, wurde der Cortex per Uart mit dem PC verbunden und die übertragenen Daten mit dem Programm X-CTU angezeigt.

Wie zu sehen ist erscheint bei Reset der Begrüßungstext Willkommen zum Inkrementalgeber: und als Zusatz wird anschließend die aktuelle Position des Inkrementalgebers übertragen.

Grenzfälle

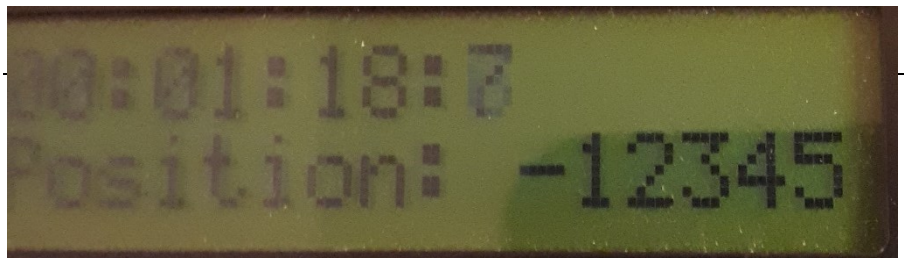
Die Variable Position wurde als integer definiert und reicht daher nur von -2147483648 bis 2147483647. Das bedeutet der Inkrementalgeber muss in diesen Rahmen bleiben. Jedoch hat das Display nur Platz für 6 Stellige zahlen.

```
298
299   position=1234567;
```



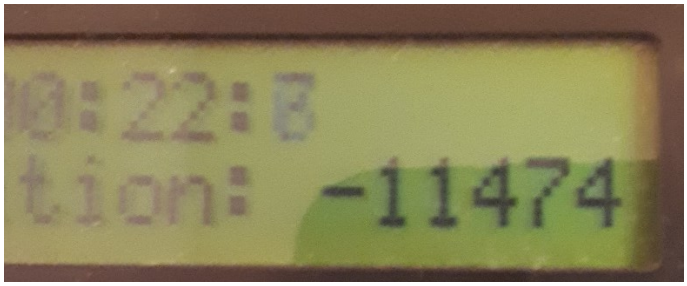
Wie zu sehen ist, wurde die 7 Stelle abgeschnitten.

```
299   position=-1234567;
```



Wie zu sehen ist, wird bei einer negativen Zahl die 6 Stelle abgeschnitten.

```
298
299   position=3147483647;
300
```



Wie zu sehen ist, wird eine Zahl die 2147483647 überschreitet, wieder negativ.

Die Variable zehntel, welche die Uhr verwendet, hätte dasselbe Problem mit dem integer Werten. Jedoch wird diese im normal Betrieb nie 863999 überscheiten können.

Es ist möglich, praktisch jedoch schwer nach zu bilden, die Richtung zu ändern während zur hälfte rauf gezählt wurde (es wird von PA1 zuerst eine negative flanke, dann von PA0 eine erwartet, um eine Linksdrehung zu erkennen und hoch zu zählen). Schafft man dies, kann hochgezählt werden, obwohl nach rechts gedreht wird oder runter, obwohl nach links gedreht wird.

7. Zeitaufwand

Patrick Steiner

Tätigkeit	**Aufwand**
Erstellung des Pflichtenheftes	10min
Erstellung des Systemdesigns	3h
Programmcodierung	8h
Testen der Software	3h
Dokumentation (Protokoll)	5h
Gesamt	**19h**

Matthias Breitseher

Tätigkeit	**Aufwand**
Erstellung des Pflichtenheftes	10h
Erstellung des Systemdesigns	3h
Programmcodierung	13h
Testen der Software	3h
Dokumentation (Protokoll)	4h
Gesamt	**23h**

8. Aufgetretene Probleme

● Aufgrund einer falschen Registereinstellung beim setzen des Taktes zu 32MHz, wurde TIM4 nur mit 8MHz versorgt. Die Prescaler und ARR werte wurden auch nicht errechnet, sondern erraten durch antasten, wodurch die Uhr nach wenigen Minuten schon Abweichungen zeigte.
●Aufgrund eines Logikfehlers beim Programmieren, bei dem die Variablen zur Bestimmung einer links/Rechtsdrehung dauernd zurückgesetzt wurden, funktionierte das hoch und runter zählen nicht.
●Die Kommunikation per uart funktionierte nicht, da das UP in amv10 von 8MHz Takt ausgegangen ist.
●Ausgabe von mehrstelligen Zahlen (01 statt 1) am Display.

9. Erkenntnisse aus der Übung

- Bedienung des Cortex M3
- Anreicherung von Kenntnissen der C-programmierung auf Mikroprozessoren
- Kenntnisse über Interrupt und Timer gesammelt
- Testen fehlerhafter Programme (Debuggen, Logikanalysator)