PIC Programmierung

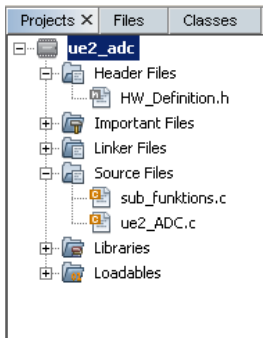
Step by Step

Übung 2

AD-Messung

Übung #2 (AD-Messung)

- **I. Neues Projekt erstellen**
  - MPLAB-X öffnen
  - File -> New Project-> Menü folgen
  - File anlegen & speichern (am besten „ue1“ kopieren und verändern)



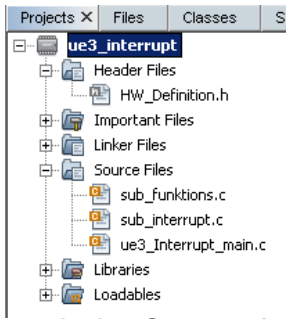
Übung #2 (AD-Messung)

- **Übung 2**
  - a) Schreiben sie folgende Funktionen
    - „void init_adc(void)“ …Initialisierung des Analog Pins
    - „char adc_8bit(char kanal)“ …8Bit AD-Messung (Input:Kanalauswahl)
    - „int adc_10bit(char kanal)“ …10Bit AD-Messung (Input:Kanalauswahl)
  - b) Schreiben sie ein Programm das die LED_rot zum Leuchten bringt, und ergänzen sie das Programm so, dass nach einem Tastendruck (S1) die Led wechselweise mit der Led_gruen blinkt.
  - c) Ergänzen sie das Programm so, dass die Blinkfrequenz mittels Poti eingestellt werden kann. Die Blinkfrequenz soll von 0.4Hz bis 5Hz einstellbar sein. (Hinweis: Die Frequenzen müssen nicht exakt sein, suchen sie einen einfachen Zusammenhang zwischen AD-Wert und Wartezeit, ohne komplexe Zwischenrechnungen, d.h. keine Fließkommaoperationen)

Übung 3
Interrupt

Übung #3 (Interrupt)

- **Neues Projekt erstellen**
- **Es soll alle 5ms ein Interrupt ausgelöst werden**
  - Lösung mit Timer0 (8Bit)
    - Prescaler im OPTION_REG einstellen (z.B.1:128)
    - 1Zähltakt dauert bei 4MHz 1µs*128=128µs
    - 39 Takte 39*128=4992us
    - InterruptFlag wird bei Überlauf gesetzt, d.h. Zähler muss beim Start auf 217 gesetzt werden




```
16  /*****************************************************************************
17      Funktion: Timerinterrupt starten
18      prescaler 1:128   @ 4MHz    --> 1us : 128(Prescaler)
19      1Takt=128usec   alle 5ms soll ein Interrupt ausgelöst werden
20      5ms/128us-->39 Zahlpulse --> Aufwärtszähler 256-39=217
21  *****************************************************************************/
22  //###############################################
23  //## reset_TMR0                 ##
24  //###############################################
25  void reset_TMR0(void)
26  {
27      TMR0 = 217;             //  movlw   .217
28                              //  movwf   TMR0
29      INTCONbits.T0IF = 0;    //  bcf     INTCON,T0IF     ;Timer0 IF löschen
30      return;
31  }//end start_tmr0_int
32
33  //###############################################
34  //## start_tmr0_int                 ##
35  //###############################################
36  void start_tmr0_int(void)
37  {
38      INTCONbits.T0IE = 0;    //Interrupt disable
39      reset_TMR0();
40      INTCONbits.T0IE = 1;
41      INTCONbits.GIE = 1;     //Interrupt enable
42      return;
43  }//end start_tmr0_int
```

Übung #3 (Interrupt)

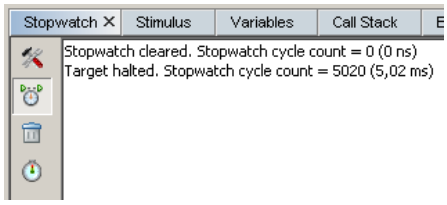
aus File „sub_interrupt.c“

```
45  //################################################
46  //## Interrupt Routine                          ##
47  //## my_ISR                                     ##
48  //################################################
49  void interrupt my_ISR(void)
50  {
51
52      NOP();                  //zum Testen (Breakpoint Sprungmarke)
53      reset_TMR0();           //neu laden, garantiert Interruptzeit
54
55      //## Hier den notwendigen Code eintragen
56
57
58      return;
59  }//end my_ISR
```

aus File „ue3_interrupt_main.c“

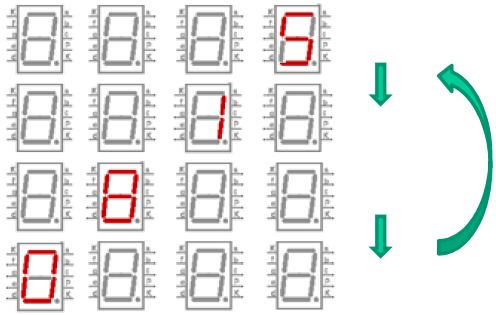
```
61  //################################################
62  //## Main
63  //################################################
64  int main(void)
65  {
66      init_pic();
67      //init_adc();       //not used
68
69      start_tmr0_int();
70
71      LED_ROT = on;
72
73      while(1);           //Endlosschleife
74
75
76  }//Ende Main
77
```

- Simulieren sie mit Hilfe des Software-Debuggers die tatsächliche Zeit zwischen 2 Interrupts



Übung #3 (Interrupt)

- **Übung 3**
  - Schreiben sie ein Programm, dass die letzten 4 Ziffern ihrer Matrikelnummer am Display ausgibt
  - Benutzen sie den Interrupt, d.h. das Hauptprogramm wird alle 5ms unterbrochen, die ISR wechselt von einem Segment auf das andere und gibt die entsprechende Zahl aus.



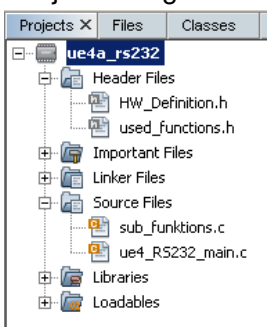
Übung 4

Serielle Kommunikation

Übung: RS232 “Funktion Initialisierung“

```
214     //################################################
215     //## init_rs232 @ 4MHz                          ##
216     //##
217     //## RX = Input / TX=Output bereits gesetzt
218     //################################################
219   void init_rs232(void)
220     {
221         // C-Code           // ASM-CODE
222         BAUDCTL = 0x08;     //Bank3
223                             //movlw b'00001000'         ;16bit Baud Rate, Autorate off
224                             //movwf BAUDCTL
225         TXSTA = 0x24;       //Bank1
226                             //movlw b'00100100'         ;8-bit,Tx_enable,asyn,high-Speed
227                             //movwf TXSTA
228         //;19200Baud --> SPBRG = 51 (0,16% Fehler)
229         //;57,6k     --> SPBRG = 16 (2% Fehler)
230         SPBRGH = 0;         //clrf  SPBRGH
231         SPBRG = 51;         //movlw .51
232                             //movwf SPBRG
233         RCSTA = 0x90;       //Bank0
234                             //movlw b'10010000'         ;SPEN=1 Continus reveive
235                             //movwf RCSTA
236
237         return;
238     }//ende init rs232
```

- Projekt anlegen



- Die Register werden lt. Datenblatt gesetzt
- Baudrate lt. Formel →Datenblatt

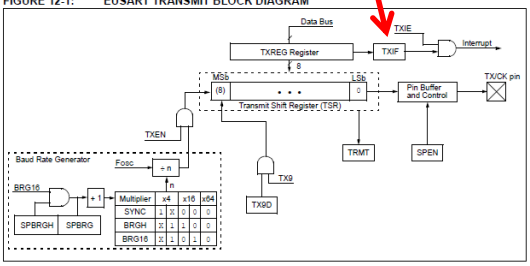
Übung : RS232 "Funktion Ausgabe"

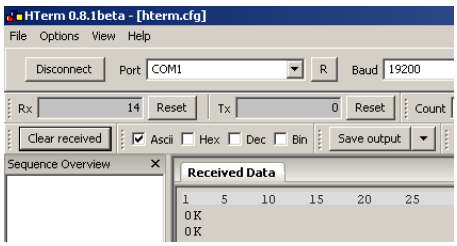
```
240  //################################################
241  //## send_zeichen (über rs232)                  ##
242  //################################################
243  void send_zeichen(char zeichen)
244  {
245      while(!PIR1bits.TXIF);  //movwf   tx_zwisch
246                              //btfss   PIR1,TXIF       ;check that buffer is empty
247                              //goto  $-1
248      TXREG = zeichen;        //movf    tx_zwisch,w
249                              //movwf   TXREG           ;transmit byte
250
251      return;
252  }//ende send_zeichen
253
254  //################################################
255  //## send_ok                                    ##
256  //################################################
257  void send_ok(void)
258  {
259      send_zeichen('O');
260      send_zeichen('K');
261      return;
262  }//ende send_ok
```

```
55   //#################################################################
56   //## Main
57   //#################################################################
58   int main(void)
59   {
60       init_pic();         // IO's Initialisierung
61       //init_adc();       // not used
62       init_rs232();       // RS232 Initialisierung
63       //start_tmr0_int(); // not used
64
65       while(1)
66           {
67           if (S1)
68               {
69               send_ok();
70               wait_xms(200);  //damit nicht zu oft gesendet wird
71               }
72           }
73
74   }//Ende Main
```

- Interrupt Flag zeigt an ob Transmit-Buffer leer ist , aber nicht ob das Byte gesendet wurde

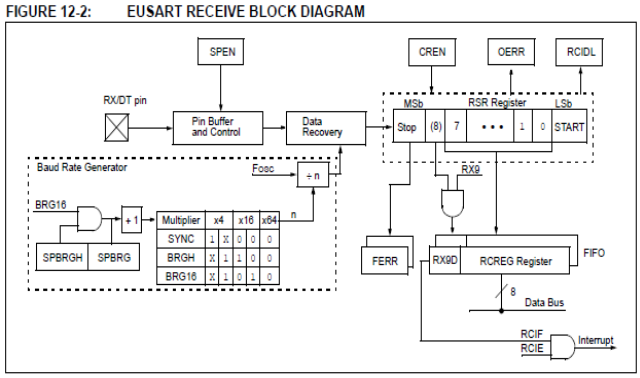
FIGURE 12-1: EUSART TRANSMIT BLOCK DIAGRAM





Übung : RS232 "Receive"

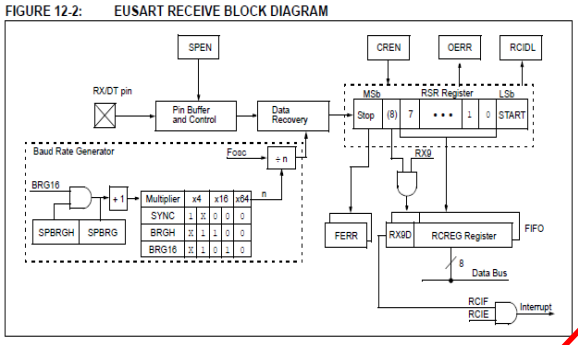
FIGURE 12-2: EUSART RECEIVE BLOCK DIAGRAM



- Receive soll per Interrupt passieren → Polling unsinnig, da CPU dauernd checken müsste, ob ein Byte empfangen wurde
- Erweiterung der „init_rs232"

```
214  //################################################
215  //## init_rs232 @ 4MHz
216  //##
217  //## RX = Input / TX=Output bereits gesetzt
218  //################################################
219  void init_rs232(void)
220  {
221      // C-Code              // ASM-CODE
222      BAUDCTL = 0x08;        //Bank3
223                             //movlw b'00001000'
224                             //movwf BAUDCTL
225      TXSTA = 0x24;          //Bank1
226                             //movlw b'00100100'
227                             //movwf TXSTA
228      //;19200Baud --> SPBRG = 51 (0,16% Fehler)
229      //;57,6k     --> SPBRG = 16 (2% Fehler)
230      SPBRGH = 0;            //clrf  SPBRGH
231      SPBRG = 51;            //movlw .51
232                             //movwf SPBRG
233      RCSTA = 0x90;          //Bank0
234                             //movlw b'10010000'
235                             //movwf RCSTA
236
237      //;Interrupt für Receive aktivieren
238      PIR1bits.RCIF = 0;  //bcf       PIR1,RCIF
239      PIE1bits.RCIE = 1;  //Bank1
240                          //bsf       PIE1,RCIE
241      INTCONbits.PEIE = 1; //Bank0
242                           //bsf       INTCON,PEIE
243      return;
244  }//ende init_rs232
```

Übung : RS232 “Receive“

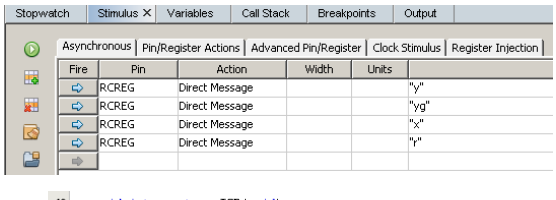


- Anpassen der Interruptroutine
- Die Leds können mit r..rot, g..grün, y..gelb ein/ausgeschaltet werden

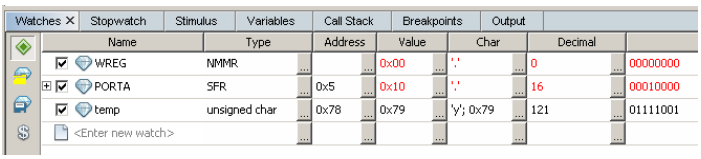
```
45  //################################################
46  //## Interrupt Routine                          ##
47  //## my_ISR                                     ##
48  //################################################
49  void interrupt my_ISR(void)
50  {
51      char temp;
52      extern char seg_pointer,zahl[4];
53
54      NOP();                  //zum Testen
55
56      if (PIR1bits.RCIF)      //### Receive Interrupt ??
57  //############## Receive Interrupt
58          {
59          temp = RCREG;       //Byte sichern
60
61          if (temp=='r') LED_ROT = ~LED_ROT;      //LED invertieren
62          if (temp=='g') LED_GRUEN = ~LED_GRUEN;  //LED invertieren
63          if (temp=='y') LED_GELB = ~LED_GELB;    //LED invertieren
64          }
65
66  //############### TMR0 Interrrupt
67      else
68          {
69          reset_TMR0();           //neu laden Interruptzeit
70
71          temp = zahl[seg_pointer];   //Zahl ausgeben
72          set_zahl(temp);
73          set_segment(seg_pointer);
74
75          seg_pointer++;          //Pointer für next ISR vorbereiten
76          if (seg_pointer>3) seg_pointer=0;
77          }
78
79      return;
80  }//end my_ISR
```


Übung : RS232 “Debuggen mit MPLAB SIM“

- Window → Simulator → Stimulus
- Breakpoint setzen
- Schrittweise debuggen
- Watch kontrollieren




```
49  void interrupt my_ISR(void)
50  {
51      char temp;
52      extern char seg_pointer,zahl[4];
53
54      NOP();                  //zum Testen
55
56      if (PIR1bits.RCIF)      //### Receive Interrupt ??
57  //############## Receive Interrupt
58          {
59          temp = RCREG;       //Byte sichern
60
61          if (temp=='r') LED_ROT = ~LED_ROT;      //LED invertieren
62          if (temp=='g') LED_GRUEN = ~LED_GRUEN;  //LED invertieren
63          if (temp=='y') LED_GELB = ~LED_GELB;    //LED invertieren
64          }
65
```

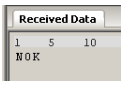


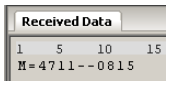
Übung #4 (RS232)

- **Übungsaufgabe 4**
  - Schreiben sie ein Programm, dass beim seriellen Empfang der Kombination „M?“ ihre Matrikelnummer an den PC zurück gibt.
  - Schon bei einem falschen Buchstaben wird sofort ein „NOK“ gesendet
  - Schreiben sie eigene Funktionen z.B. „send_mtr“ & „send_nok“
  - Hinweis: LookupTabelle verwenden

```
const char my_text[] = {"M=4711--0815"};  // 12 Zeichen
```

  - Testen sie das Programm mit dem Terminalprogramm „hterm“ und Simulator

Bsp „x“ → 

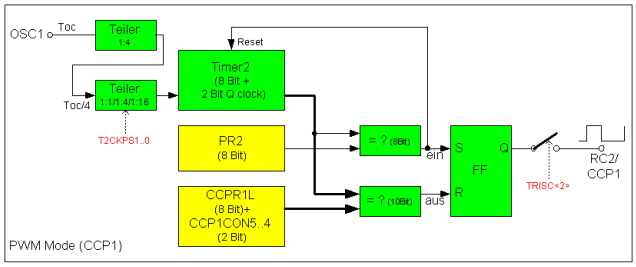
Bsp „M?“ → 

Übung 5

PWM

Übung: PWM (Grundlagen)

Im PWM-Mode wird der Timer2 mit einem festen Takt gespeist. Er beginnt bei 0 zu zählen. Der Zählwert des Timer2 wird ständig mit den Werten in den Registern CCPR1L und PR2 verglichen. Erreicht der Timer2 den Wert von CCPR1L, dann wird der Ausgang CCP1 auf Low-Pegel gesetzt. Erreicht der Timer2 den Wert vom PR2, dann wird der Ausgang CCP1 auf High-Pegel gesetzt und der Timer2 auf 0 zurückgesetzt. Der Zyklus beginnt von vorn.
Die Periode der Schwingung hängt also neben dem Timer-Takt von PR2 ab. Das Tastverhältnis bestimmt das Verhältnis von PR2 und CCPR1L. Ist CCPR1L größer als PR2, dann bleibt der Ausgang CCP1 immer auf High.



Eigentlich sind sowohl der Timer2, wie auch PR2 und CCPR1L sind nur 8-bittig. Um eine 10-bittige PWM-Auflösung zu erreichen, wird CCPR1L um zwei zusätzliche LSB erweitert, die die Bits 5 und 4 von CCP1CON sind. Zum Vergleich werden neben den 8 Timer2-Bits zwei weitere LSB-Bits herangezogen, die aus dem Vorteiler des Timer2 oder aus dem Toc-Teiler (4:1) stammen.

Folglich lässt sich die Periode der erzeugten Rechteckschwingung nur mit 8 Bit einstellen, während das Tastverhältnis mit bis zu 10-Bit eingestellt werden kann.

Übung: PWM (Grundlagen)

Um den PWM-Mode zu nutzen muss man

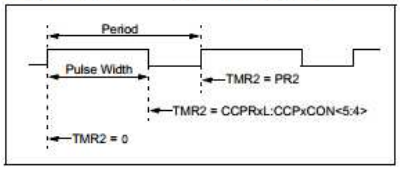
- die Periode der Rechteckschwingung einstellen
- dabei auch den Timer2 mit dem richtigen Takt versorgen und einschalten
- das Pulsverhältnis (duty-cycle) einstellen
- das CCP-Pin zum Ausgang machen
- den PWM-Mode aktivieren

Die Periode der Rechteckschwingung hängt ab von

- dem Takt des PIC
- der Einstellung des Timer2-Vorteilers
- dem Wert im Register **PR2**

Das Pulsverhältnis wird mit dem Register **CCPR1L** und zwei Bits in **CCP1CON** eingestellt.

FIGURE 11-4: CCP PWM OUTPUT



Übung: PWM "Funktion Initialisierung"

```
217  void init_pwm(void)
218  {
219      //1. Disable PWM-PIN
220      TRISCbits.TRISC2 = 1;       //input
221      //2. PWM Periode    10kHz->100us (mit TMR2 1:1) -->PR2=99
222      PR2 = 99;
223      //3. CCP1CON Register setzen
224      CCP1CON = 0b00001100;       //P1A, PWM-Mode active low
225      //4. PWM Duty Cycle 0%
226      CCPR1L = 0x00;
227      //5. Config TMR2
228      PIR1bits.TMR2IF = 0;
229      T2CON = 0x00;           //Pre 1:1
230      T2CONbits.TMR2ON = 1;   //Timer on
231      //6. Warten auf ersten überlauf --> Pin=Out
232      while(PIR1bits.TMR2IF);
233      TRISCbits.TRISC2 = 0;       //Output
234      return;
235  }
236  //################################################
237  //## Set Duty
238  //#
239  //# mit PR2=99, TMR2=1:1
240  //# duty=0      ton=0us  toff=100us  -> 0%
241  //# duty=99     ton=99us toff=1us    -> 99%
242  //# duty>99     nie aus = 100%
243  //################################################
244  void set_proz_duty(unsigned char duty)
245  {
246      //if (duty>100)  duty = 100;    //max 0-100 erl
247      CCP1CON = duty;
248      return;
249  }
```

EQUATION 11-1: PWM PERIOD

$$PWM\ Period = [(PR2) + 1] \bullet 4 \bullet TOSC \bullet (TMR2\ Prescale\ Value)$$

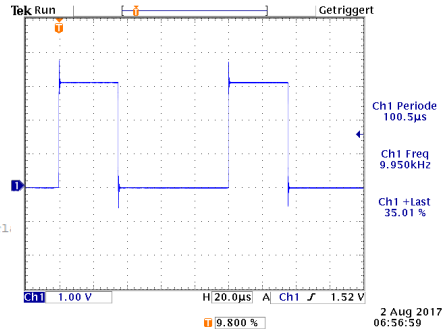
Note: TOSC = 1/FOSC

EQUATION 11-2: PULSE WIDTH

$$Pulse\ Width = (CCPRxL:CCPxCON<5:4>) \bullet TOSC \bullet (TMR2\ Prescale\ Value)$$

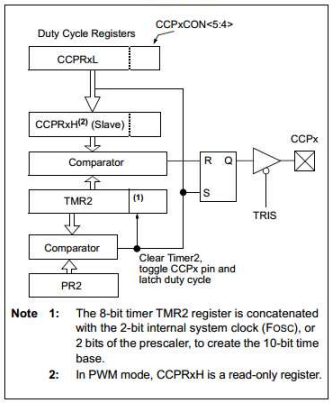
EQUATION 11-3: DUTY CYCLE RATIO

$$Duty\ Cycle\ Ratio = \frac{(CCPRxL:CCPxCON<5:4>)}{4(PR2 + 1)}$$



- Die Register werden lt. Datenblatt gesetzt

FIGURE 11-3: SIMPLIFIED PWM BLOCK DIAGRAM



Note 1: The 8-bit timer TMR2 register is concatenated with the 2-bit internal system clock (FOSC), or 2 bits of the prescaler, to create the 10-bit time base.
2: In PWM mode, CCPRxH is a read-only register.

Übung #5 (PWM)

- **Übungsaufgabe 5**
  - Schreiben sie ein Programm, dass mittels Poti die Helligkeit der Led regelt.