Mikroprozessorpraktikum WS 2011/12
Aufgabenkomplex: 7

Teilnehmer:

Marco Träger, Matr. 4130515
Alexander Steen, Matr. 4357549

Gruppe: Freitag, Arbeitsplatz: HWP 1

A 7.1 UART Senden

7.1.1 Folgende Informationen zur Lösung der Aufgabe:

- UART1 an den Portleitungen P3.6 (TxD) und P3.7 (RxD) wird genutzt
- Controller arbeitet mit 7,3728MHz DCOCLK-Takt
- SMCLK-Taktquelle für Baudratengenerator
- Baudrate 115200 Bit/sek
- Datenformat ist 8,N,1
- Initialisierung der UART erfolgt mit der `init_UART1()` aus init.c

Vervollständigung der Funktion `init_UART1()` aus show: init.c

```
void init_UART1(void)
{
  P3SEL |= (0x03) << 6;          // USART RX und TX
                                 // dem Modul zuweise
  U1CTL |= SWRST;                // Reset
    // Char-Format 8N1
  U1CTL &= ~PENA;                // No Parity Bit
  U1CTL |= CHAR;                 // 8 Datenbit
  U1CTL &= ~SPB;                 // 1 Stoppbit
    // Char-Format ende
  U1TCTL |= (SSEL0 + SSEL1);     // Taktquelle SMCLK
  U1BR0 = 64;                    // Teiler Low-Teil, da 7372800/64
                                 // da wir so ca die Baudrate von
                                 // 115200 Bit/s erreichen
  U1BR1 = 0;                     // Teiler High-Teil
  U1MCTL = 0;                    // Modulationskontrolle
  ME2 |= (UTXE1 + URXE1);        // Enable USART1 TXD/RXD
  U1CTL &= ~SWRST;               // Reset
  IE2 |= (UTXIE1 + URXIE1);      // Enable Interrupt
}
```

7.1.2 Entwickeln Sie ein kleines Programm, welches aus Tastendruck P1.0 das Zeichen '?' über die UART zum PC schickt. Auf dem PC muß zum Empfang des Zeichens ein Terminalprogramm mit den entsprechenden Parametern gestartet werden.

Wir machen den Taster P1.0 interruptfähig.

```
void init712() {
    // Taste 1 input, IO, Interrupt, edge
    P1DIR &= ~(0x01);
    P1SEL &= ~(0x01);
    P1IE |= (0x01);
    P1IES &= ~(0x01);
}
```

Nun wird immer wenn der Taster P1.0 gerückt oder losgelassen wird ein Interrupt erzeugt auf den in der ISR PORT1 reagiert werden kann. Um nur auf Tastendrücke zu reagieren prüfen wir als erstes ob der Taster gedrückt wurde. Falls ja senden wir das Zeichen '?'. Um ein Prellen des Schalters zu verhindern warten wir noch eine kurze Zeit am Ende der Routine und beenden sie dann mit dem Zurücksetzten des Interruptflags.

Ansonsten ist in diesem Programm nichts zu tun, daher lassen wir die Hauptschleife leer.

```
#pragma vector = PORT1_VECTOR
__interrupt void PORT1 (void) {

  if((P1IN & 0x01)) {
    // Taster P1.0 gedrueckt
    sendchar('?');
  }
```

```
  wait(5000);
  CLEAR(P1IFG, 0xFF);
}

void aufgabe712() {
  // skip
}
```

Das senden eines Zeichens kann durch eine einfache Funktion realisiert werden. Wir warten bis der Sendepuffer U1TCTL der UART-Einheit 1 leer ist. Anschießend schreiben wir das zu sendene Zeichen in den Sendepuffer dieser Einheit. Das Zeichen wird nun automatisch versendet.

```
void sendchar(char c) {
  // wenn TXEPT == 0 dann werden noch Daten transportiert. Wir
      warten bis
  // keine mehr transportiert werden.
  while((U1TCTL & TXEPT)== 0x00);
  U1TXBUF = c;
}
```

7.1.3 Schicken Sie bei jedem Tastendruck eine Zeichenkette 'Hallo World' zum PC.

In der Initialisierung machen wir, analog zu 7.1.2, den Taster P1.0 interruptfähig.

Wir senden nun das Wort 'Hallo World' anstatt eines einzelnen Zeichens.

```
#pragma vector = PORT1_VECTOR
__interrupt void PORT1 (void) {

  if((P1IN & 0x01)) {
    // Taster P1.0 gedrueckt
    send("Hello World");
  }

  wait(5000);
  CLEAR(P1IFG, 0xFF);
}
```

Das senden eines Wortes (einer Zeichenkette) kann dabei durch ein successives Aufrufen der Sende-Funktion für ein einzelnes Zeichen realisiert werden, da die Sendefunktion für ein einzelnes Zeichen bereits wartet bis der Sendepuffer frei ist bevor das Zeichen gesendet wird.

```
void send(char* str, int index, int length) {
  int i;
  int end = index + length;
  for(i = index; i < end; i++) {
    sendchar(str[i]);
  }
}
```

A 7.2 UART Empfang

7.2.1 Initialisieren Sie die UART1 in der Form, dass ein empfangenes Zeichen einen Interrupt auslöst.

- wird das Zeichen E empfangen, schaltet LED P4.1 ein
- wird das Zeichen A empfangen, schaltet LED P4.1 aus
- wird das /CR Zeichen empfangen, toggelt LED P4.2

Wir aktivieren wir den UTXIFG1 Interrupt, um auf Daten die von USART1 empfangen werden zu reagieren und bereiten die nötigen LEDs vor.

```
void init722() {
    LED1OFF;LED2OFF;LED3OFF;LEDOFF;

    IE2 = URXIE1; // Emfangsinterrupt aktivieren
    // LEDS vorbereiten
    P4DIR |= (0x07);
    P4SEL &= ~(0x07);
    P4OUT |= 0x07;
}
```

Die ISR ließt das empfangene Zeichen auf dem `USART1` Emfangsbuffer `U1RXBUF` ein. Anhand einer simplen Fallunterscheidung werden nun die LEDs entsprechen an oder aus geschalten.

```
#pragma vector = USART1RX_VECTOR
__interrupt void USART_Receive (void)
  {
    switch(U1RXBUF) {
      case 'E':
        LED2ON;
        break;
      case 'A':
        LED2OFF;
        break;
      case '\r':
        P4OUT ^= 0x04;
        break;
    }
  }
```

7.2.2 Über ein Terminalprogramm am PC soll eine Zeichenkette eingegeben und mit CR+LF abgeschlossen werden. Das Terminalprogramm überträgt die Zeichenkette zum Controller. Dieser soll die Zeichenkette im Interrupt empfangen, die Länge der Zeichenkette bestimmen und zum PC zurückschicken.

Wir aktivieren wir den UTXIFG1 Interrupt, um auf Daten die von USART1 empfangen werden zu reagieren.

```
void init722() {
    LED1OFF;LED2OFF;LED3OFF;LEDOFF;

    IE2 = URXIE1; // Emfangsinterrupt aktivieren
}
```

In der ISR lesen wir abermals das empfangene Zeichen. Wir zählen jedes mal wenn wir ein Zeichen empfangen einer Zähler eins höher um uns zu merken wie viele Zeichen wir bereits empfangen haben.

Um ein aufeinander folgendes '\r\n' zu bemerken, merken wir uns in einer zusätzlichen Kontrollvariable ob wir ein '\r' lesen. Wenn wir ein '\n' lesen prüfen wir mit Hilfe der Kontrollvariable, ob wir als letzen ein '\r' gelesen haben, falls ja haben wir ein Zeichen zu viel gezählt. Wir ziehen dieses von der ermittelten Anzahl ab und übertragen die Anzahl der gelesenen Zeichen. Anschließend setzen wir sowohl die Anzahl der gelesenen Zeichen als auch die Kontrollvariable für '\r' zurück.

Wann immer wir ein Zeichen lesen das nicht '\r' ist, setzen wir die Kontrollvariable für '\r' konsequenterweise zurück.

```
unsigned char count722 = 0;
bool waslastcharCR = 0;

// diese Implementierung kann nur Strings der Laenge 127 emfangen
#pragma vector = USART1RX_VECTOR
__interrupt void USART_Receive (void)
  {
    switch(U1RXBUF) {
      case '\r':
```

```
        waslastcharCR = 1;
        count722++;
        break;
      case '\n':
        if(waslastcharCR) {
          count722--;   // CR wurde mitgezaehlt, wieder abziehen
          sendchar(count722);
          count722 = 0;
        }
        waslastcharCR = 0;
        break;
      default:
        count722++;
        waslastcharCR = 0;
        break;
    }
  }
```

Der Einfachheit halber haben wir uns dafür entschlossen maximal bis zu 127 Zeichen zählen zu können. Diese Beschränkung entsteht da wir mit einem Zeichen nur maximal einen 7-Bit-Char übertragen können. Um größere Zahlen übertragen zu können, müsste man mehrere Zeichen hintereinander versenden.

A 7.3 Sensordaten

7.3.1 Entwickeln Sie ein Programm das zyklisch die Uhrzeit und die Werte des SHT11-Sensors als ASCII-Zeichenkette auf der seriellen Schnittstelle ausgibt.

Wir haben uns für einen Zyklus von einer Sekunde entschieden, daher initialisieren wir einen Timerinterrupt jede Sekunde anlog zu Aufgabe 6.

```
void init731() {
    LED1OFF;LED2OFF;LED3OFF;LEDOFF;

    // Timer setzen
    TBCCR0 = (unsigned int)32 768; // Taktanzahl fuer eine
        Sekunde
    // Taktquelle ACLK ist Flag TBSSEL_1, Countmethod ist MC_1
    TBCTL &= ~(TBSSEL_3 + MC_3 + ID_3);//reset
    TBCTL |= (TBSSEL_1 + MC_1 + ID_0 );//set

    TBCCTL0 |= CCIE ; // Interrupt freigeben
}
```

Wenn immer die ISR des Timers aufgerufen wird ist eine Sekunde vergangen, daher können wir einfach in einem Zähler die vergangenen Sekunden zählen (`t731`). Wir konvertieren die Anzahl der Sekunden in das Format `hh:mm:ss`.

Die Sensor-Daten des Sensors SHT11 können über die Funktion `SHT11_Read_Sensor()` abgefragt werden. Die Funktion schreibt die aktuellen Daten des Sensors in die Register `temp_char` und `humi_char`. Daher können wir nach dem Aufrufen der Funktion einfach auf diese zugreifen.

Nun hängen wir die konvertierte vergangene Zeit und die aktuellen Sensor-Daten einfach in einem String aneinander und senden diese mit der Funktion `send()` (die bereits weiter oben vorgestellt wurde).

```
unsigned int t731 = 0;

#pragma vector = TIMERB0_VECTOR
__interrupt void TIMER (void) {
  char zeitausgabe[100];
  unsigned int s; unsigned int m; unsigned int h;

  t731++;

  s = t731%60u;
```

```
  m = (t731/60u)%60u;
  h = (t731/3600u)%60u;

  SHT11_Read_Sensor();
  sprintf(zeitausgabe,"%02u:%02u:%02u, %s, %s\n", h, m, s,
      temp_char, humi_char);
  send(zeitausgabe);
}
```