1.5. Weitere Hinweise zum Laborversuch

- Wie kann eine Warteschleife mit einer vorgegebenen Zeitdauer in Software realisiert werden?

Laborversuch 2: UART

Name	*Vorname*	*Studien-gang*	*Matr.-Nr.*	

Gruppennummer:

2.1. Ziele des Laborversuchs

➔ Heutzutage werden UART Schnittstellen noch in Geräten verwendet, in denen Funktionen mittels des AT-Befehlssatzes angesteuert werden. Beispielsweise werden AT-Befehle in Modems und GSM-Modulen für die Einwahl in ein Telefonnetz, für die Wahlwiederholung sowie für weitere Einstellungen in …(Telefongeräten)… (Lautsprecherkontrolle, Auswahl des Betriebsmodus, Überwachung der Verbindungszeit, …). In diesem Laborversuch werden Sende- und Empfangsfunktionen eines UART Moduls implementiert und getestet. Verschiedene Übertragungsmodi werden berücksichtigt.

➔ Weiterhin wird man lernen, wie eine in Assembler geschriebene Funktion in C aufgerufen werden kann. Hierbei soll man insbesondere auf die Übergabe von Funktionsargumenten achten. Für ARM Prozessoren werden die in einer höheren Programmiersprache (wie C z.B.) geschriebenen Programme gemäß den allgemeinen APCS (ARM Procedure Call Standard) Vereinbarungen in Assembler übersetzt. Ein Verständnis solcher Vereinbarungen kann unter anderen eine Optimierung eines in C geschriebenen Programms ermöglichen.

➔ Die automatische Messung der Datenrate (Auto-Baud) eines UART Signals wird auch durchgeführt. Eine solche Messung kann dazu dienen, die softwareseitig vorgenommenen Einstellungen der Zeitparameter zu überprüfen.

2.2. Vorbemerkungen zur Vorbereitung und Durchführung des Laborversuchs

➔ Die Vorbereitung des Laborversuchs erfolgt anhand eines taktgenauen Simulationsmodells des Mikrocontrollers. Lösungen zu den im Laborversuch gestellten Übungen werden mit dem Simulator in Form von Hausaufgaben entwickelt und getestet.

➔ Die mit dem Simulator entwickelten Lösungen werden von einem Betreuer ausgewertet. **Eine positive Auswertung gilt als notwendige Bedingung für die weitere Durchführung des Laborversuchs**. Während der Durchführung des Laborversuchs werden Lösungen, die mit dem Simulator entwickelt wurden, auf die Ziel-Hardware portiert und anschließend getestet.

➔ Als Hardwareplattform wird das **EVB (Evaluation Board) MCB2300** von der Firma Keil verwendet. Das MCB2300 EVB (siehe **Abbildung 2**) unterstützt ARM-Mikrocontroller der LPC23xx Familie von der Firma NXP (Ehemalige Halbleiter-Sparte von Philips). Für diesen Laborversuch sind die ausgewählten MCB EVBs mit einem LPC2368 oder LPC2378 Mikrocontroller bestückt und bieten zahlreiche Schnittstellen (CAN, UART, Ethernet, USB, …) sowie eine LCD-Anzeige mit 2 Zeilen à 16 Zeichen an.

In diesem Laborversuch werden folgende Schnittstellen des MCB2300 EVB verwendet:

- COM0 und COM1 PORTs (jeweils für UART0 und UART1)
- LCD-Anzeige

Obwohl der Mikrocontroller 4 UART Module (UART0, UART1, UART2 und UART3) beinhaltet, werden nur die 2 Module UART0 und UART1 in diesem Laborversuch verwendet, da das ausgewählte Testboard (EVB MCB2300) Anschlüsse mit Sub-D Verbindern nur für diese 2 UART Module anbietet.

2.3. Vorbereitung

Folgende Assembler-Programme stehen Ihnen zur Verfügung:

- ***UART_init.s***
- ***UART_PutChar.s***

Das Assembler-Programm ***UART_init.s*** kann das Peripherie-Modul UART0 oder UART1 konfigurieren. Um dieses Assembler-Programm sinngemäß verwenden zu können, müssen die Prozessor-Register R0, R1, R2 und R3 wie folgt geladen werden:

- R0 = UART_PortNum
  UART_PortNum = 0 oder 1, je nachdem, ob UART0 oder UART1 konfiguriert werden soll.
- R1 = U_DL
  U_DL = (256*UxDLM) + UxDLL
  Die Bits 7 bis 0 von U_DL werden in das Register UxDLL geladen, während die Bits 15 bis 8 ins das Register UxDLM geschrieben werden.
- R2 = UART_Mode
  UART_Mode beinhaltet den für das Register UxLCR vorgesehenen Inhalt.
- R3 = (INT_ENABLE << 8) | FIFO_Mode
  Der für das Register UxIER vorgesehene Wert wird in die Bitstellen 15 bis 8 vom Register R3 geladen, während der Inhalt vom Register UxFCR mit den Bits 7 bis 0 vorgegeben wird.

(Man beachte: x = 0 oder 1 in diesem Laborversuch)

Werfen Sie bitte einen Blick in den Assembler Code hinein, um den Konfigurationsvorgang verstehen zu können. Achten Sie insbesondere auf die Verwendung des DLAB Bit im Register UxLCR.

Mit dem Assembler-Programm ***UART_PutChar.s*** lässt sich ein ASCII Zeichen mit dem Peripherie-Modul UART0 oder UART1 senden. Vor der Ausführung dieses Programms sollen die Prozessor-Register R0 und R1 wie folgt gefüllt werden:

- R0 = UART_PortNum (0 oder 1)
- R1 = Zeichen (Char)
  Der Bytewert des sendenden Zeichen (engl. Character) solle in das Register R1 geladen werden.

In diesem Programm wird das Bit 5 (THRE Bit) im Register UxLSR solange geprüft, ob es gesetzt ist. Wenn dieses Bit gesetzt ist, wird der Bytewert ins Register UxTHR geschrieben und anschließend durch den TX Pin vom UART Modul gesendet.

➔ Übung L2.1

Testen Sie die Funktionskorrektheit des Assembler Programms ***UART_PutChar.s*** mit einer Zeichenkette, die Sie im Assembler Code vorgeben. Die Zeichenkette soll mit dem Bytewert 0 enden. Folglich dürfen **keine weiteren Zeichen nach diesem Bytewert 0** gesendet werden. Die Übung soll mit beiden UART Modulen (UART0 und UART1) durchgeführt werden. Hierbei sollen folgende Parameter für den UART Sendevorgang verwendet werden:

- Datenrate 9600 Bit/s
- 8 Nutzdatenbits pro Sendung, keine Parität, 1 Stopp Bit
- Interrupts ausgeschaltet
- FIFO aktiv, Trigger level = 1

Das zu schreibende Assembler-Programm kann wie folgt aussehen:

```
1     AREA UART_TX_Test, CODE, READONLY
2     INCLUDE LPC2368_asm.inc
3     EXPORT start
4     IMPORT UART_init
5     IMPORT UART_PutChar
6
7  start
8     ;Configure UART
9     LDR  R0, ...
10    LDR  R1, ...
11    LDR  R2, ...
12    LDR  R3, ...
13    BL  UART_init
14
15    |
16
17      PUT YOUR CODE Hier
18
19    |
20
21 Done
22    B   Done
23
24    AREA    String_Block, DATA, READONLY
25 MyString  DCB  "Life is good.",0
26
27    END
```

Die Bezeichnung (Label) **MyString** stellt die Adresse der zu sendenden Zeichenkette dar.

Selbstverständlich sollen die Assembler Programme ***UART_init.s*** und ***UART_PutChar.s*** in das Projekt eingebunden werden.

Abbildung 50: Schablonendatei für das Assembler-Programm zum Testen der UART Sendefunktion.

Prüfen Sie bitte die Konfiguration des ausgewählten UART Moduls im entsprechenden Fenster des Debugger Werkzeugs. Achten Sie darauf, dass die eingestellte Datenrate leicht verfehlt werden kann, da die ***Fractional Part*** Werte (MULVAL und DIVADDVAL) nicht verwendet werden.

Menü:Peripherals →UART →UART0 oder UART1

Im View Fenster des verwendeten UART Moduls können Sie gesendeten Zeichen sehen.

Menü:View →Serial Windows →UART#1 (für UART0 !) oder UART#2 (für UART1 !)

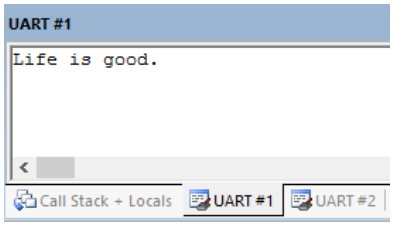

Abbildung 51: Ausgabefenster vom UART 0 Modul

➔ Übung L2.2

Schreiben Sie ein Assembler Programm ***UART_GetChar.s***, mit dem ASCII Zeichen über das UART Modul UART0 oder UART1 empfangen werden. Für dieses Programm gilt die Voraussetzung, dass die Auswahl des UART Moduls mit dem Register R0 erfolgt.

Also: R0 = PortNum (0 oder 1) vor Ausführung der Funktion

In diesem Programm soll die Bezeichnung (Label) **UART_GetChar** vor dem ersten Assembler Befehl (und natürlich nach den anfänglichen Direktiven) stehen. Ähnlich wie mit den Assembler Programmen ***UART_init.s*** und ***UART_PutChar.s*** soll dieses Programms mit einem Sprungbefehl BL (Branch with Link) aus einer Anwendung aufrufbar sein. Daher sollen die Inhalte der Register, die vom zu schreibenden Programm geändert werden, mit einem STMFD (Store Multiple/Full Descending) Befehl gerettet werden. Am Ende des Programms sollen die geretteten Inhalte mit einem LDMFD (Load Multiple/Full Descending) wieder zurückgeschrieben werden. Eine genaue Vorstellung bezüglich der STMFD und LDMFD Befehle lässt sich durch einen Blick in die Dateien ***UART_init.s*** und ***UART_PutChar.s*** gewinnen.

Das Assembler Programm ***UART_GetChar.s*** soll nur einmal prüfen, ob das Bit RDR im Register UxLSR prüfen. Falls dieses Bit gesetzt ist, nimmt man an, dass ein gültiges Zeichen im Register UxRBR vorliegt. Das Programm soll dieses Zeichen lesen und ins Prozessor-Register R0 schreiben. Wenn das Bit RDR im Register UxLSR zurückgesetzt ist, soll der Wert 0xFFFFFFFF (= -1) ins Prozessor-Register R0 geladen werden.

Das Assembler Programm ***UART_GetChar.s*** soll nun mit den anderen Assembler Programmen ***UART_init.s*** und ***UART_PutChar.s*** getestet werden. Hierfür soll ein Assembler Programm **UART_LoopBack_ASM.s** geschrieben werden, das Zeichen über eine UART

Schnittstelle empfängt und dann durch den Anschluss der gleichen UART Schnittstelle zurücksendet. Somit das Programm so lange warten, bis ein Zeichen empfangen wird und unmittelbar das empfangene Zeichen zurücksenden. Im Simulator können die ini-Datei U0_terminal_emulation.ini oder U1_terminal_emulation.ini (jeweils fürs UART0 oder UART1 Modul) verwenden. Diese ini-Datei wird ein Toolbox Menü auftauchen lassen. Wenn Sie auf den Button *U0_terminal_emulation* oder *U1_terminal_emulation* klicken, werden alle Buchstaben in alphabetischer Reihenfolge jeweils in das UART0 oder UART1 Modul eingespeist.

Benutzen Sie bitte die im **Verzeichnis L2_Uebung2** vorhandenen Schablonendateien für die Assembler Programme ***UART_GetChar.s*** und **UART_LoopBack_ASM.s.** Mit dem Assembler soll die Nummer des Anschlusses angepasst werden, wenn das Modul UART0 oder UART1 ausgewählt wird.

Im Ausgabefenster (siehe **Abbildung 52**) des ausgewählten UART Moduls können Sie prüfen, ob Ihr Assembler Programm die empfangenen Zeichen zurückgibt.

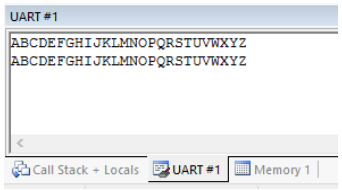

Abbildung 52: Ausgabe des UART_LoopBack_ASM.s Programms mit dem UART0 Modul