Laborversuch 2: UART

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 wie beispielsweise Lautsprecherkontrolle, Auswahl des Betriebsmodus und 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 im Fokus stehen. Eine solche Messung kann dazu dienen, die softwareseitig vorgenommenen Einstellungen der Zeitparameter zu überprüfen.

2.2. Vorbemerkungen zur Vorbereitung und Durchführung des Laborversuchs

In diesem Laborversuch werden folgende Schnittstellen vom LandTiger EVB verwendet:

- ❖ COM0 und COM1 PORTs (jeweils für UART0 und UART2)
- ❖ LCD Display Modul
- ❖ INT0 Button

Obwohl der Mikrocontroller 4 UART Module (UART0, UART1, UART2 und UART3) beinhaltet, werden nur die 2 Module UART0 und UART2 in diesem Laborversuch verwendet, da das ausgewählte Testboard (LandTiger EVB) 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 UART2 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 2, je nachdem, ob UART0 oder UART2 konfiguriert werden soll.
- R1 = UxDL
  UxDL = (256*UxDLM) + UxDLL
  Die Bits 7 bis 0 von U_DL werden in das Register UxDLL geladen, während die Bits 15 bis 8 in das Register UxDLM geschrieben werden.
- R2 = (UxFDR << 8) | UxLCR (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 2 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 UART2 senden. Vor der Ausführung dieses Programms sollen die Prozessor-Register R0 und R1 wie folgt gefüllt werden:

- R0 = UART_PortNum (0 oder 2)
- R1 = Zeichen (Char)
  Der Bytewert des zu sendenden Zeichens (engl. Character) soll 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: UART Sendefunktion testen (Programmiersprache Assembler) (Sim)

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 (Null-Terminierung). Folglich dürfen **keine weiteren Zeichen nach diesem Bytewert 0** gesendet werden. Die Übung soll mit beiden UART Modulen (UART0 und UART2) durchgeführt werden. Hierbei sollen folgende Parameter für den UART Sendevorgang verwendet werden:

- Datenrate 115200 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:

```
    THUMB ; Directive indicating the use of UAL
    AREA Code1, CODE, READONLY, ALIGN=4

    INCLUDE LPC1768.inc
    IMPORT UART_init
    IMPORT UART_PutChar

    EXPORT __main
    ENTRY
__main  PROC
    ; Configure UART
    LDR  R0, = ...       ; UART_PortNum  0 or 2
    LDR  R1, = ...    ; U_DL  = (U_DLM << 8) | (U_DLL)
    LDR  R2, =0x3     ; UART_Mode (LCR):  DLAB 0  Set Break 0  Stick Parity 0  Even Parity Select 0
                      ;Parity Enable 0  Number of Stop Bits 0  Word Length Select 11
    LDR  r3, =0x4003    ; Int Enable (IIR)+FIFO_Mode (FCR) ;TX FIFO Reset 1;RX FIFO Reset 1;FIFO Enable 1

    ;R0 = UART_PortNum, R1 = U_DL , R2 = UART_Mode, R3 = (INT_ENABLE << 8) | FIFO_Mode
    ; 8 Nutzdatenbits pro Sendung, keine Parität, 1 Stopp Bit ; Interrupts ausgeschaltet
    ; FIFO aktiv, Trigger level = 1

    MOV r4,r0           ; Copy of PortNum
    BL  UART_init
    LDR r5, =MyString   ;

Read_String             ; MainLoop
    ;
    ; PUT YOUR CODE HERE
    ;
FinishLoop
    B   .   ; End Of Program (Sprung auf sich selbst)

    ENDP

    AREA    String_Block, DATA, READONLY
MyString  DCB  "Life is good.",10,0     ; 10 = LF (Line Feed) for new line, 0 = End of String

    END
```

Abbildung 48: Schablonendatei für das Assembler-Programm zum Testen der UART Sendefunktion.

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.

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, falls die zur Berechnung der Datenrate benötigte Division keine Ganzzahl geliefert hat.

Menü: Peripherals → UART → UART0 oder UART2

Nachdem die Datei UART_Test_TX.s mit Assembler Befehlen ergänzt wurde, soll sie in das Projekt eingefügt.

Prüfen Sie Ihren Code und starten Sie eine Debug Sitzung.

Führen Sie den Code aus.

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

Menü: View → Serial Windows → UART#1 *(für UART**0** !)*

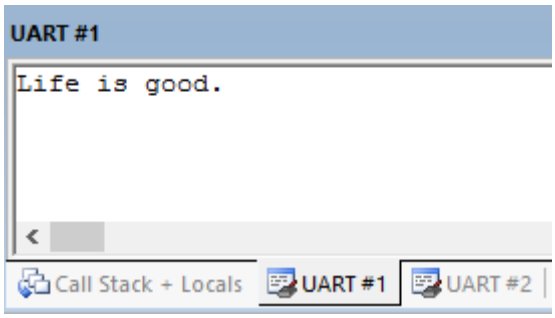

Abbildung 49: Ausgabefenster vom UART 0 Modul

Hinweis: Diese Ausgabemöglichkeit ist **nur für UART0 verfügbar!**
Im Menü sind zwar weitere Einträge UART#2 und UART#3 enthalten, die aber keine Funktion erfüllen.

➔ Übung L2-2: UART Empfangsfunktion in Assembler schreiben und testen (Sim)

Schreiben Sie ein Assembler Programm ***UART_GetChar.s***, mit dem ASCII Zeichen über das UART Modul UART0 oder UART2 empfangen werden. Für dieses Programm gilt die Voraussetzung, dass die Auswahl des UART Moduls mit dem Register R0 erfolgt.

Also: R0 = PortNum (0 oder 2) 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 Programm 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 gesetzt ist. 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 Sie die ini-Datei ***U0+U2_terminal_emulation.ini*** (für die UART0 oder UART2 Module) verwenden. Diese ini-Datei wird ein Toolbox Menü auftauchen lassen. Wenn Sie auf den Button *U0_terminal_emulation* oder *U2_terminal_emulation* klicken, werden alle Buchstaben in alphabetischer Reihenfolge jeweils in das UART0 oder UART2 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 UART2 ausgewählt wird.

Im Ausgabefenster (siehe **Abbildung 50**) des ausgewählten UART Moduls können Sie prüfen, ob Ihr Assembler Programm die empfangenen Zeichen zurückgibt.

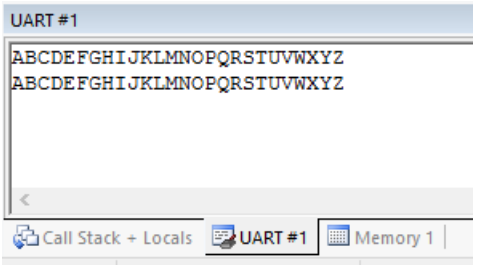

Abbildung 50: Ausgabe des UART_LoopBack_ASM.s Programms mit dem UART0 Modul

➔ Übung L2-3: Vorhandene Assembler-Funktionen in C gemäß APCS Regeln aufrufen(Sim)

Für ARM Prozessoren werden Compiler so geschrieben, dass die Verwendung von Prozessor-Registern gemäß der APCS (ARM Procedure Call Standard) Vereinbarungen erfolgt. Die Betrachtung der Register nach APCS wird in **Abbildung 51** veranschaulicht.

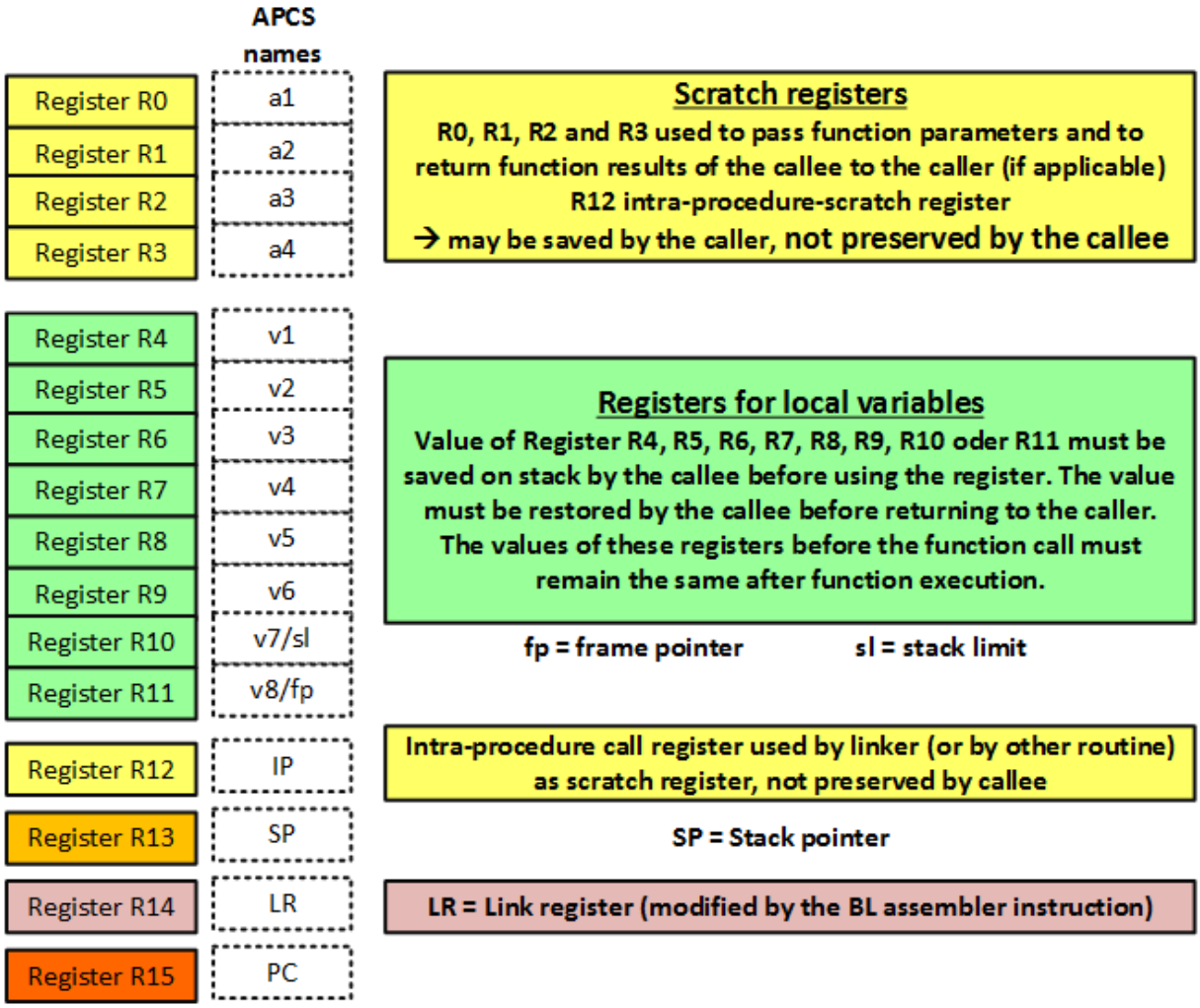

Abbildung 51: Verwendung von Prozessor-Registern nach APCS

Die Register R0, R1, R2 und R3 dienen zur Übergabe von Parameterwerten zwischen einer aufrufenden Funktion (*engl. Caller*) und einer aufgerufenen Funktion (*engl. Callee*). Vor dem Aufruf einer Funktion wird die aufrufende Funktion (Caller) die entsprechenden Parameterwerte in die Register R0, R1, R2 und R3 laden. Daher soll eine C Funktion bestenfalls bis zu 4 Parameter aufweisen. Für eine Funktion mit mehr als 5 Parametern wird der Stack auch zur Übergabe von Parameterwerten verwendet. Dadurch wird die Übergabe von Funktionsargumenten aufwendiger und somit zu einer Verschlechterung der Performanz beitragen. Es empfiehlt sich daher, maximal 4 Parameter für eine C Funktion vorzusehen, wenn diese Funktion für einen ARM Prozessor kompiliert werden soll.

Weiterhin dienen die Register R0, R1, R2 und R3 zur Rückgabe von Ergebnissen einer Funktion. Wenn der Rückgabewert beispielsweise vom Typ ***integer*** ist, soll die aufgerufene Funktion nach deren Ausführung den zurückzugebenden Wert in das Register R0 schreiben. Für einen Rückgabewert vom Typ ***double*** werden die Register R0 und R1 benötigt.

Die Register R4 bis R11 werden nach den APCS Regeln zur Speicherung von lokalen Variablen einer aufgerufenen Funktion (Callee) verwendet. Daher soll die aufgerufene Funktion den Inhalt von jedem verwendeten Register R4, R5, R6, R7, R8, R9, R10 oder R11 vor der Ausführung retten und nach der Ausführung wiederherstellen.

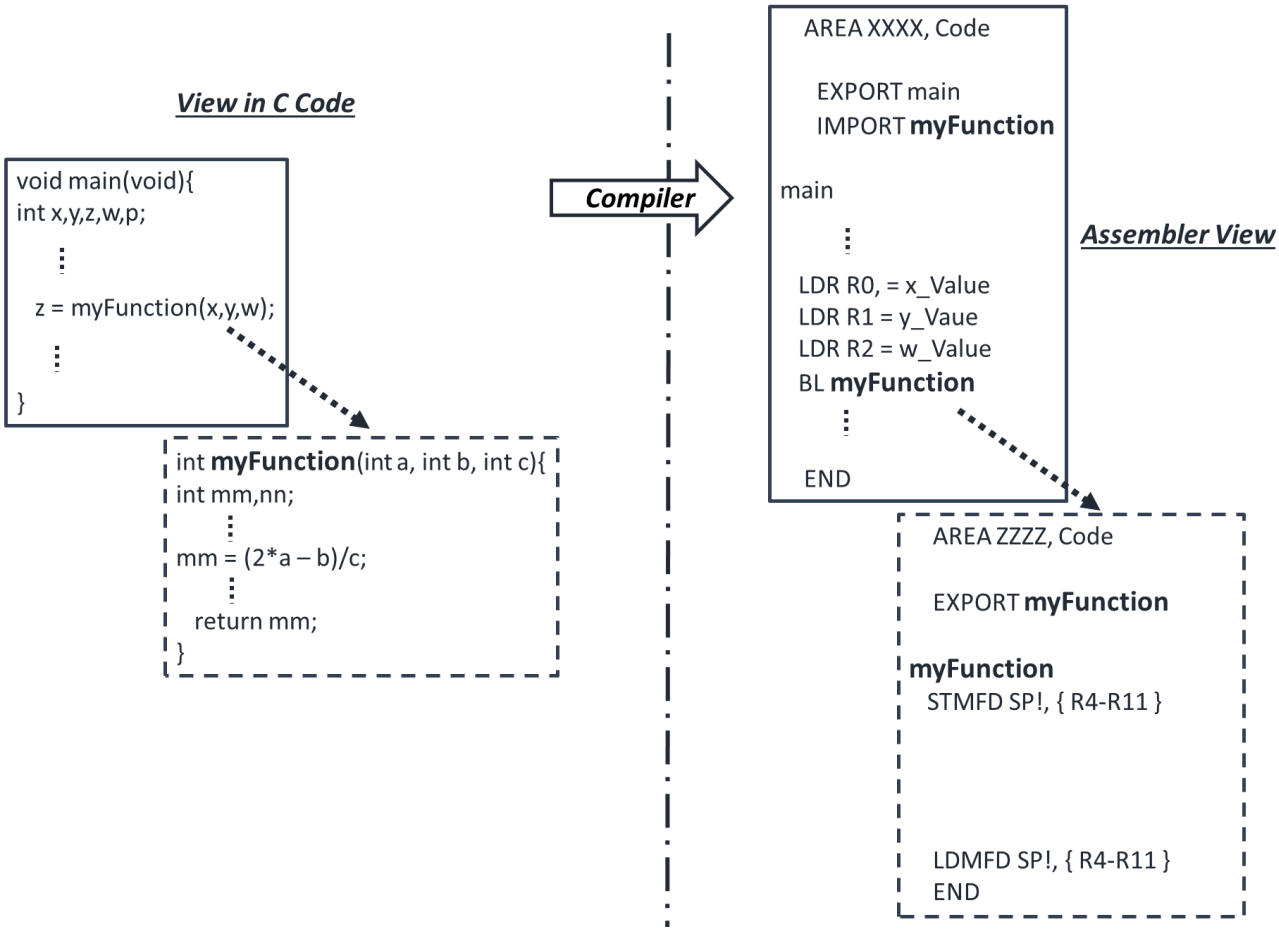

Abbildung 52: Anwendungsbeispiel von APCS Regeln

Die Assembler Funktion **UART_init.s** kann nun wie folgt in einem C Programm aufgerufen werden. Diese Funktion setzt voraus, dass die Register R0 bis R3 belegt sind. Daher kann man davon ausgehen, dass 4 Parameter benötigt werden.

➔ Auszug aus der Assembler Datei

```
;R0 = UART_PortNum, R1 = U_DL , R2 = UART_Mode, R3 = (INT_ENABLE << 8) | FIFO_Mode
```

In ein C Programm lässt sich die Assembler Funktion **UART_init.s** mit dem folgenden Funktionsprototyp einbinden:

```
extern void UART_init(int UART_PortNum,int U_DL,int UART_Mode,int INT_EN_FIFO_Mode;
```

Danach kann die Funktion UART_init mit passenden Argumenten in einem C Programm aufgerufen werden.

Beispiel: UART_init(0, 94, 3, 3); //UART Port 0, Baudrate 9600 Bit/s, 8 Bits, No Parity, 1
//Stop Bit, FIFO enabled, RX FIFO cleared

Schreiben Sie nun ein C Programm ***UART_LoopBack_C.c***, in dem die Assembler Funktionen ***UART_init.s, UART_GetChar.s*** und ***UART_PutChar.s*** aufgerufen werden. Das C Programm soll auf ein Zeichen an einer UART Schnittstelle solange warten, bis ein Zeichen empfangen wird. Das empfangene Zeichen soll unmittelbar mit dem gleichen UART Modul zurückgesendet werden. Anschließend wird das Programm auf ein weiteres Zeichen warten, um dieses wieder zurückzusenden. Diese dabei entstandene Schleife soll wiederholend durchgeführt werden. Ähnlich wie in Übung L2.2 können Sie die ini-Datei U0+U2_terminal_emulation.ini (jeweils fürs UART0 oder UART2 Modul) im Simulator verwenden. Im Verzeichnis L2_Uebung3 befindet sich eine Schablonendatei UART_LoopBack_C.c, die Sie ergänzen können. Vergessen Sie nicht Ihre Datei ***UART_GetChar.s*** in das Verzeichnis L2_Uebung3 zu kopieren.

2.4. Versuchsdurchführung im Labor

➔ Übung L2-4: UART-basierter Datenaustausch zwischen dem PC und dem Mikrocontroller via das HTERM Programm (HW)

In den vorherigen Übungen wurden UART Datentransfers im Polling Modus durchgeführt. Im Polling Modus fragt ein Software Programm den Wert bestimmter Status Bits wiederholt ab und läuft nur weiter, wenn die abgefragten Werte den Auftritt des erwarteten Ereignisses signalisieren. Für praktische Anwendungen ist der UART Polling Modus nicht empfehlenswert, da der Mikrocontroller über keinen ausreichenden Zeitpuffer für andere Aufgaben verfügt. Daher werden wir uns hier den Interrupt Betriebsmodus eines UART Moduls anschauen.

Empfangsseitig ist ein RX Interrupt unverzichtbar, da ein UART Modul die Ankunft neuer Daten über eine UART Leitung nicht vorahnen kann. Ein aktives Warten auf die Ankunft neuer Daten kann daher als verlorene Zeit betrachtet werden.

Senderseitig stellt ein TX Interrupt einen Mehrwert dar, wenn die Anzahl der zu sendenden Bytes größer als die eingestellte TX FIFO Tiefe ist. Die ersten Bytes werden in der Regel ohne Interrupt gesendet. Dabei wird der TX Interrupt Modus für die nachfolgenden Bytes freigeschaltet. Bei den UART Modulen des LPC1768 Mikrocontrollers sollen zuerst 2 Bytes in TX FIFO ohne Interrupt geschrieben werden, bevor der TX Interrupt Modus aktiv wird. Falls ein Mikrocontroller immer auf das Ende eines Sendevorgangs eines Byte warten soll, wird er viel Zeit verlieren. Man erinnere sich daran, dass die Datenrate einer UART Datenübertragung viel niedriger als der Datendurchsatz in einem Peripherie-Modul eines Mikrocontrollers ist. Die UART Datenrate liegt typischerweise zwischen 9600 Bit/s und 115200 Bit/s (also: zwischen 1200 Byte/s und 14400 Byte/s), während ein Peripherie-Modul Bytes intern beispielsweise mit einer Frequenz von 25 MHz (also: mindestens 25.000.000 Byte/s) verarbeiten kann.

Ein TX Interrupt für ein UART Modul soll am besten wieder ausgeschaltet werden, nachdem das letzte Byte eines vorherigen Datenblocks gesendet wurde. Solange kein neuer zu sendender Datenblock vorliegt, soll ein UART TX Interrupt gesperrt bleiben.

Im Verzeichnis **L2-4** befinden sich Dateien, die Sie für den Interrupt Betrieb eines UART Moduls verwenden können. Zurzeit ist die Interrupt Fähigkeit nur für das Modul UART0 konfiguriert.

Im UART Modul ist die RX Interrupt Leitung standardmäßig freigeschaltet. Jede empfangene Zeichenkette wird in den Zeilen 3 bis 8 am Display Modul des LandTiger EVB ausgegeben. Mit der ausgewählten Schriftgröße können 20 Zeichen pro Zeile ausgegeben werden. Der ausgegebene Text wird 10 Sekunden lang angezeigt. Diesen **Zeitabstand** können Sie in Software nach Ihrer Präferenz ändern. Ein längerer Text wird in Teile von maximal 20 Zeichen zerlegt, so dass diese Teile der Reihe nach jeweils nach vorgegebenem Zeitabstand

am LCD Display angezeigt werden. Die Folge der angezeigten Teile einer Zeichenkette bleibt solange erhalten, bis eine neue Zeichenkette empfangen wird. Eine neue Nachricht muss mit **Character 0x0A (LineFeed)** beendet sein, damit dies zur Anzeige führt.

Wenn eine aus Buchstaben in alphabetischer Reihenfolge bestehende Zeichenkette empfangen wird, führt dies zu einer Anzeige wie in Abbildung 53 dargestellt.

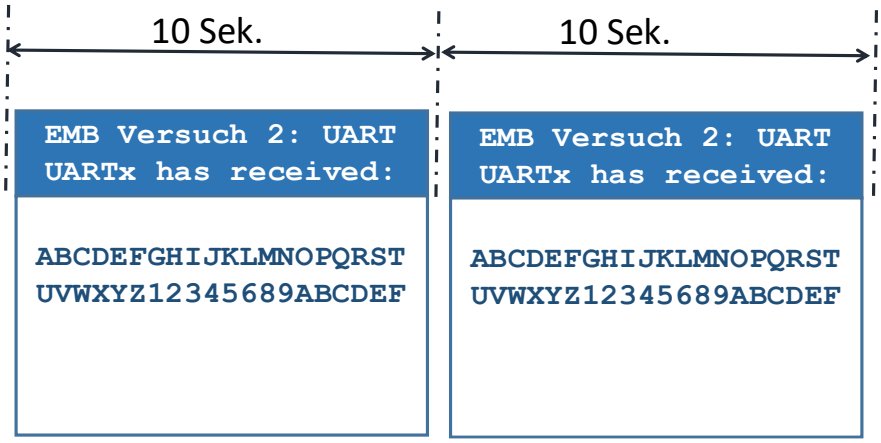

Abbildung 53: Ausgabe am LCD Display nach UART Empfang per Interrupt

Ein UART Sendevorgang wird mit dem Taster INT0 ausgelöst. Dabei werden die zwei ersten Zeichen gesendet. Die weiteren Zeichen werden per TX Interrupt gesendet. Nach dem letzten Zeichen wird die UART TX Interrupt Leitung gesperrt. Die zu sendende Zeichenkette befindet sich in der Datei ***MyText.h***.

Ihre Aufgabe besteht nun darin, die Konfiguration des Interrupt Controllers (NVIC) sowie die Initialisierung des Timer 0 Moduls. Hierfür können eine Funktion

void init_NVIC(void)

in C schreiben, die benötigten Interrupts freischaltet. Prioritätsstufe vom Timer0 Interrupt soll auf 3 gesetzt werden, während UARTx Interrupts eine Prioritätsstufe von 2 aufweisen soll.

Vergessen Sie nicht die entsprechenden Interrupt Leitungen im Interrupt Controller freizuschalten.

(Erinnerung: Je höher die einzutragende Zahl ist, umso niedriger ist die Prioritätsstufe.)

Weiterhin sollen Sie auch die Funktion

void init_Timer0_for_UART(void)

in C schreiben. Diese Funktion soll das Timer 0 Modul so konfigurieren, dass eine LCD Ausgabe mindestens stehen soll. Wenn der auszugebende Text mehr als 120 Zeichen umfasst, wird er so zerlegt, dass ein Teil des Texts mindestens für eine für Timer 0

einstellbare Zeitdauer angezeigt wird. Diese Zeitdauer kann beispielsweise 10 Sekunden oder zum Testen auch eine kürzere Zeit sein.

Denken Sie auch daran, das von Ihnen geschriebene Assembler Programm **UART_GetChar.s** ins Verzeichnis **L2-4** zu kopieren.

Im Simulator können Sie **zuerst** prüfen, ob die Konfiguration und Initialisierung in den Funktionen ***init_NVIC( )*** und ***init_Timer0_for_UART( )*** einwandfrei laufen. Diese zwei Funktionen können am Ende der Datei **UART_api.c** eingefügt werden.

Mithilfe eines Haltepunkts („Breakpoint“) in einer ISR (Interrupt Service Routine) können Sie prüfen, ob Interrupts wie erwartet ausgelöst werden. Als ini-Datei können Sie die Datei **L2_Uebung4.ini** auswählen. Mit dem entsprechenden Menü stehen folgende Möglichkeiten zur Auswahl:

- Taster_toggle → Emulation vom INT0 Taster.
- U0_Terminal_Emu.
- U1_Terminal_Emu.

Die Buttons bieten die gleiche Funktionalität wie in der vorherigen Übung.

➔ **Das Terminalprogramm HTerm wird im Labor eingeführt.**

➔ Übung L2-5: Hardware vs. Software Messung der Datenrate einer UART Datenübertragung (Sim)

Als Entwickler von eingebetteten Systemen muss man in der Lage sein, die Korrektheit von in Software vorgenommenen Einstellungen zu prüfen. In der vorgesehenen Versuchsdurchführung (siehe Absatz 2.4) soll ein UART Modul so konfiguriert werden, dass die Baudrate empfangener Daten automatisch gemessen wird. Hierbei werden die in Hardware errechneten Werte in den Registern UxDLL und UxDLM automatisch gespeichert.

Diese automatisch errechneten Werte sollen in der abschließenden Laboraufgabe L2-6 mithilfe einer Input Capture Funktion eines Timer Moduls überprüft werden.

In dieser Übung soll nun die Messung der UART Baudrate anhand einer Input Capture Funktion eines Timer Moduls vorbereitet werden, indem die minimale Zeitdauer zwischen 2 aufeinanderfolgenden Flanken (d.h. zwischen einer fallenden Flanke und der nächsten steigenden Flanke bzw. zwischen einer steigenden Flanke und der nächsten fallenden Flanke) eines vorgegebenen digitalen Signals gemessen wird.

Mithilfe der ini-Datei **Terminal+Auto Baud CAP2.1(P0.5).ini** wird ein Testsignal im Simulator generiert und durch den Pin CAP2.1 (P0.5) in den Mikrocontroller eingespeist. Mit einem Klick **Launch_P0.5_Signal** auf diesen Button wird ein Signal am Pin CAP2.1 (P0.5) generiert.

Schreiben Sie eine Funktion ***init_Timer2( )*** zur Initialisierung des Timer 2 Moduls, um eine Input Capture Funktionalität am Pin CAP2.1 (P0.5) zu ermöglichen. Schreiben Sie auch eine ISR (Interrupt Service Routine) ***TIMER2_IRQHandler*** (), die eine Messung des minimalen Zeitabstandes zwischen zwei aufeinanderfolgenden Flanken durchführt. Aus dem minimalen Zeitabstand soll ebenfalls die gleichbedeutende Baudrate berechnet werden. Testen Ihre Funktionen ***TIMER2_IRQHandler*** und ***init_Timer2( )*** im Simulator, indem Sie nur den Button **Launch_P0.4_Signal** verwenden, um ein Signal zur Durchführung der Messung zu erzeugen.

Denken Sie auch daran, die Funktion ***init_NVIC( )*** hinsichtlich der Abarbeitung von Timer 2 Interrupt zu erweitern. Die Prioritätsstufe soll auf 2 gesetzt werden.

Im Logic Analyzer Tool des Simulators sollen Sie nun die minimale Zeitspanne zwischen zwei aufeinanderfolgenden Flanken mit dem Mauszeiger messen. Vergleichen Sie nun die im Logic Analyzer Tool gemessene Zeitspanne mit dem in Software anhand der Input Capture Funktion ermittelten minimalen Zeitwert. Wenn die Abweichung zwischen den zwei Werten kleiner als 5 % liegt, sind Sie mit dieser Übung fertig und Ihre Input Capture Funktion darf in der Versuchsdurchführung verwendet werden.

➔ Aufgabe L2-6 Autobaud, Senden+Empfangen (HW)

Im Labor wird der Mikrocontroller NXP LPC1768 durch ein UART Modul Daten mit dem PC austauschen. Auf dem PC wird das Terminalprogramm **Hterm** für die serielle Schnittstelle gestartet. In der grafischen Oberfläche dieses Programms kann man unter anderem den Port (z.B. COM1 PC-seitig), die Baudrate und die Parität der PC-Schnittstelle bestimmen.

Achten Sie darauf, dass die PC-Einstellungen im **Hterm** Terminalprogramm mit den Einstellungen im Mikrocontroller übereinstimmen.

In der Versuchsdurchführung soll das Terminalprogramm **Hterm** Zeichen mit einer vorgegebenen Datenraten an den Mikrocontroller senden, so dass zwei Messungen der UART Datenrate im Mikrocontroller erfolgen:

- Eine automatische Messung durch das gewählte UART Modul
- Eine Messung mithilfe einer Input Capture Funktion

Um diese Messung vereinfachen zu können, soll der PC verschiedene Bitströme senden, die vorzugsweise eine Bitfolge 1 0 1 und/oder 0 1 0 aufweisen. Hierbei kann beispielsweise das Zeichen ‚**U**‘ vorzugsweise am Anfang einer Zeichenkette übertragen werden, da die ASCII Codierung dieses Zeichens 0x55 (= Dezimal 85) die obige Anforderung erfüllt.

Ein UART Modul im NXP LPC1768 kann im Auto-Baud Modus die Datenraten eines empfangenen Bitstroms messen und die (von der UART Hardware) automatisch gemessene Datenrate übernehmen. Hierbei werden diese in Hardware errechneten Werte in den Registern UxDLL und UxDLM automatisch gespeichert.

Die **Auto-Baud Messung kann nicht im Simulator durchgeführt werden**. Daher wird der Auto-Baud Modus nur mit der Ziel-Hardware im Labor getestet.

Eine Auto-Baud Messung lässt sich starten, nachdem der Wert 0x05 in das Register UxACR geschrieben wird und Daten per UART empfangen werden.

- UxACR Bit0 = 1 → Zeigt an, dass eine Auto-Baud Messung für den nächsten Empfang eines ASCII-Zeichens freigeschaltet ist. Nach der ersten Durchführung einer automatischen Messung der Baudrate wird dieses Bit zurückgesetzt.

- UxACR Bit1 = 0 → UART befindet sich in Mode 0 (d.h. Bitzeit über Start-Bit und LS Bit errechnet).
- UxACR Bit2 = 1 → Mit diesem Bit erfolgt eine neue automatische (Hardware) Messung der Baudrate nach Ablauf einer in Hardware vorgegebenen Frist.

Die per Autobaud ermittelte Baudrate kann im UART-Registerfenster in
> Peripherals > UART > UART0
unter Divisor Latch / Baudrate: … angezeigt werden. Hierfür ist es aber nötig, den Programmlauf zu stoppen und manuell im Line Control Register das DLAB-Bit zu setzen.

Für eine **Messung der UART Baudrate mit dem Input Capture Pin CAP2.1 (P0.5)** soll man eine Verbindung mit dem zu betrachtenden RX Pin des ausgewählten UART Moduls realisieren. Hierbei lässt sich die benötigte Verbindung mittels der Pinout Nachbildung ermöglichen.

Binden Sie die Funktionen ***TIMER2_IRQHandler*** und ***init_Timer2( )*** in das neue Projekt ein, so dass die UART Bit-Zeit kontinuierlich als minimale Zeitspanne zwischen zwei aufeinanderfolgenden Flanken gemessen wird. Unter der Voraussetzung, dass mindestens eine Bitfolge 0 1 0 oder 1 0 1 durch den ausgewählten UART Port übertragen wurde, lässt sich die gemessene minimale Zeitspanne als eine UART Bit-Zeit interpretieren. Daher kann diese minimale Zeitspanne zur Berechnung der UART Datenrate dienen. Sie sollen die Funktion ***TIMER2_IRQHandler*** anpassen, so dass eine Berechnung der UART Datenrate anhand der gemessenen minimalen Zeitspanne erfolgt.

Für 3 verschiedene UART Datenraten (z.B. 9600 Bit/s, 19200 Bit/s und 38400 Bit/s) sowie für eine andere Datenrate Ihrer Wahl sollen die Ergebnisse der 2 Messverfahren (d.h., Messung der Input Capture Funktion und UART automatische Messung) verglichen werden. Die Ergebnisse müssen gut miteinander korrelieren.