h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

MIKROPROZESSORPRAKTIKUM

SS2022

Termin 5

Programmierung für eingebettete Systeme: Pointer, Peripherie, USART, SWI

Name, Vorname	Matrikelnummer	Anmerkungen
Datum	Raster (z.B. Mi3x)	Testat/Datum

Legende: V:Vorbereitung, D: Durchführung, P: Protokoll/Dokumentation, T: Testat

Arbeitsverzeichnins:

Kopieren Sie sich aus dem „sftp://stxxyyyy@userv-shell.fbi.h-da.de/share/LabDisk/MI/“ den Ordner mpsSS2022. Dort finden Sie zu jedem Termin vorgegebene Dateien.

Lernziele:

Die Kenntnis der Basistechnologie zur Implementierung einer Schnittstelle zwischen Anwendungsprogrammen und Betriebssystemen. Die Bedeutung und Anwendung Supervisor- und User-Mode. Möglichkeiten der Ein- und Ausgabe von Zeichen über eine serielle Schnittstelle. Vorbereiten von Testszenarien bei Zugriff auf ein Zielsystem im Labor.

Aufgabenstellung:

Der SWI Befehl führt zu einer Ausnahmebehandlung im Prozessor die mit einem Wechsel in den Supervisor Mode verbunden ist. Dem SWI Befehl kann beim Aufruf ein bis zu 24 Bit großer Wert übergeben werden, der in einem SWI Handler dazu benutzt werden kann eine gewünschte Funktion/Aktion auszuwählen.

In der Aufgabe soll ein SWI-Handler gezeigt/genutzt werden, um eine Trennung des Low Level IOs vom Anwendungsprogramm zu erreichen. Dazu sind die Funktionen/Unterprogramme die im Supervisor-Mode ausgeführt werden in Assembler implementiert/zu implementieren. Mit dem Debugger ist die Funktionsweise des SWI-Handlers zu untersuchen.

Aufgabe 1:

Nehmen Sie die zur Verfügung gestellten Dateien in ein neues Projekt auf, testen, diskutieren, verstehen und dokumentieren Sie diese.

Ermitteln Sie welche Parameter in der Terminalemulation (z.B. minicom) einzustellen sind, damit eine Kommunikation über die serielle Schnittstelle funktionieren wird.

_____

Beschreiben Sie den Unterschied der Unterprogramme/Prozeduren init_ser() und initswi().

_____

Was passiert oder könnte passieren, wenn Sie direkt nach den Ausgaben von LF und CR durch putch(0xa) und putcswi(0xd) noch weitere Zeichen mit putch('a') und/oder putcswi('b') (in Echtzeit) auf die gegebenen Weise ausgeben und warum?

_____

Die erzeugten Ausgaben von CR (Wagenrücklauf) und LF (Zeilenvorschub) sollten auf einem Terminal an der seriellen Schnittstelle zu sehen sein. Verwenden Sie in einer separaten Shell das Programm ***„minicom“*** als Terminalersatz. Hinweis: „minicom“ muss auf dem zum Entwicklungssystem zugehörigen PC gestartet werden.

Aufgabe 2:

Erklären Sie den Code des SWI-Handlers (siehe swi.c/swi.S). Debuggen Sie Ihr Programm und lokalisieren Sie die Umschaltung vom User Mode in den Supervisor Mode und zurück.

Woran erkennen Sie, in welchem Mode sich der Prozessor befindet?

_____

An welcher Stelle wird der Supervisor Mode verlassen?

_____

Aufgabe 3:

Die Initialisierungs-Routine *void initsswi(void)* und die IO-Funktionen char putcswi(char) und char getcswi() ermöglichen es, die vorhandenen in C geschriebenen Programme *int init_ser(void)*, *char putch(char)* und *char getch(void)* im Supervisor Modus auszuführen. Die benötigten Rückgabewerte der in C-geschriebenen Funktionen werden entsprechend durch gereicht. Also auch wenn die serielle Schnittstelle nicht bereit wäre, liegt die Entscheidung beim Nutzer, ob auf die gerade nicht sende- oder empfangsbereite Schnittstelle gewartet werden soll.

> Wie könnten/müssten die Aufrufe der Funktionen *char putcswi(char)*, *char getcswi(void)*, *char putch(char)* und *char getch(void)* verpackt werden, damit gewartet wird, bis ein Zeichen gesendet bzw. empfangen wurde. Testen Sie Ihre Lösung und weisen Sie die Funktionalität nach.

Aufgabe 4:

Können Sie den fehlenden in Assembler zu implementierenden Programmteil für int putsswi(char *) (siehe ***ser_io.S***) so ergänzen, dass auch ein String im Supervisor Mode auf die serielle Schnittstelle ausgegeben wird?

> ***int putsswi(char *)*** Ausgabe eines nullterminierten Strings und Rückgabe der Anzahl der ausgegebenen Zeichen.

Sie können auch die in C geschriebene Routine int puts(char *) zur Ausgabe von Strings nutzen.

Hinweis: Für den Termin 6 benötigen Sie für die serielle Ein- und Ausgaben keine privilegierten Rechte.

Aufgabe 5:

Entwickeln und **testen** Sie für Ihr Projekt eine möglichst effektive und speichersparende Routine, mit der Sie eine vorzeichenbehaftete Integerzahl in einen String zur Ausgabe auf die Konsole (serielle Schnittstelle) wandeln.

> Mit welchen Werten werden Sie Ihre Routine testen, um die volle/richtige Funktionalität nachzuweisen?

Aufgabe 6:

Erstellen Sie zu diesem Termin ein Protokoll mit den Lösungen zu den Aufgaben und Ihren Erkenntnissen. Das Protokoll sollen Sie zu den nächsten Terminen vorlegen können. Denken Sie daran, dass zum letzten (sechsten Termin) auch eine Dokumentation zu Ihrem Projekt (Funktions- und Programmbeschreibungen, Installationsanleitung, Inbetriebnahme, Benutzerhandbuch) vorgelegt werden soll.

```
// FileName:  Termin5Aufgabe1.c

// Programmrahmen zur Aufgabe Termin5
// Aufgabe 1
//***********************************
//
// von:
// vom:
// letzte Aenderung:
// von:

int main(void)
{
        char Zeichen = 0;
// Serielle initialisieren
        init_ser();
        initsswi();
// CR und LF auf das Terminal ausgeben
        putch (0xa);
        putcswi (0xd);
// ein evtl. vorhandenes Zeichen von der seriellen (Eingabe über Terminal) abholen und ausgeben
        Zeichen=getch();
        if (Zeichen !=0)
          while(!(putch(Zeichen)));
// String ausgeben
        puts("Hallo! \n");
        putsswi(„swiHallo!\n“);

        return 0;
}
```

Vorschlag eines Makefile zu Termin5 SS2021

```
# Quelldatei
FILE = Termin5Aufgabe1
# Toolchain
TOOLCHAIN = arm-eb63-elf
# Compiler
COMPILER = gcc
# Linker/Binder
LINKER = ld
# Pfad zur libgcc.a
# fuer RA_MPS_SS2021.ova
LIBGCC = /opt/arm-eb63-elf/lib/gcc/arm-eb63-elf/4.4.1/libgcc.a
# Debugger
DEBUGGER = insight
# Optimierungsstufe
OPTI = 0

# Bauen
all:
# uebersetzen der Quelldatei
	$(TOOLCHAIN)-$(COMPILER) -c -g -O$(OPTI) $(FILE).c -I ../h

# Erzeugen der Assemblerdateien aus den Quelldateien
	$(TOOLCHAIN)-$(COMPILER) -S -O$(OPTI) $(FILE).c -I ../h
	$(TOOLCHAIN)-$(COMPILER) -S -O$(OPTI) seriell.c -I ../h
	$(TOOLCHAIN)-$(COMPILER) -S -O$(OPTI) swi.c -I ../h

# Erzeugen der benoetitgen Objektdateien
# eigener SoftWareInterrupt-Handler
	$(TOOLCHAIN)-$(COMPILER) -c -g -O$(OPTI) swi.c -o swi.o -I ../h
	$(TOOLCHAIN)-$(COMPILER) -c -g -O$(OPTI) seriell.c -o seriell.o -I ../h
	$(TOOLCHAIN)-$(COMPILER) -c -g -O$(OPTI) ser_io.S -o ser_io.o -I ../h
	$(TOOLCHAIN)-$(COMPILER) -c -g -O$(OPTI) ../boot/boot_ice.S -o boot_ice.o -I ../h

# Binden fuer die RAM-Version
	$(TOOLCHAIN)-$(LINKER) -Ttext 0x02000000 -O$(OPTI) boot_ice.o swi.o seriell.o ser_io.o $(FILE).o -o $(FILE).elf $
(LIBGCC)
#	$(TOOLCHAIN)-$(LINKER) -Ttext 0x02000000 -O$(OPTI) boot_ice.o swi.o seriell.o $(FILE).o -o $(FILE).elf $(LIBGCC)

# Debugger starten
debug:
	$(TOOLCHAIN)-$(DEBUGGER) $(FILE).elf

# Aufraeumen
clean:
	rm -f *.o
	rm -f *.s
	rm -f *.elf
	rm -f *.rom

# BDI2000 zuruecksetzen funktioniert auf den Laborechnern MI-2014-x
bdireset:
	bash bdi_reset
```

```
/*-------------------------------------------------------------------------------
@ File Name:          seriell.c
@ Object:  Grundfunktionen der seriellen Schnittstelle
@                     int init_ser(); char putch(char); char getch();
@
@ Autor:              M.Pester
@ Datum: 04.12.2007
@-------------------------------------------------------------------------------*/
#include "../h/pmc.h"
#include "../h/pio.h"
#include "../h/usart.h"

int init_ser(void);
char putch(char);
char getch(void);

#define    DEFAULT_BAUD 38400
#define    CLOCK_SPEED 25000000
//CD =  (CLOCK_SPEED / (16*(DEFAULT_BAUD))    // 25MHz / ( 16 * 38400) = 40.69  -> 41 -> 0x29
#define    CD 0x29

// Initialisiert die serielle Schnittstelle USART0
int init_ser()
{
        StructPIO* piobaseA = PIOA_BASE;
        StructPMC* pmcbase  = PMC_BASE;
        StructUSART* usartbase0 = USART0;

        pmcbase->PMC_PCER = 0x4;            // Clock für US0 einschalten
        piobaseA->PIO_PDR = 0x18000;        // US0 TxD und RxD
        usartbase0->US_CR = 0xa0;           // TxD und RxD disable
        usartbase0->US_BRGR = CD;           // Baud Rate Generator Register
        usartbase0->US_MR = 0x8c0;          // Keine Parität, 8 Bit, MCKI
        usartbase0->US_CR = 0x50;           // TxD und RxD enable

        return 0;
}

// Gibt wenn möglich ein Zeichen auf die serielle Schnittstelle aus
// und liefert das Zeichen wieder zurück
// wenn eine Ausgabe nicht möglich war wird eine 0 zurück geliefert
char putch(char Zeichen)
{
        StructUSART* usartbase0 = USART0;

        if( usartbase0->US_CSR & US_TXRDY )
        {
                usartbase0->US_THR = Zeichen;
        }
        else
        {
                Zeichen = 0;
        }
        return Zeichen;
}

// Gibt entweder ein empfangenes Zeichen oder eine 0 zurück
char getch(void)
{
        StructUSART* usartbase0 = USART0;
        char Zeichen;

        if( usartbase0->US_CSR & US_RXRDY )
                Zeichen = usartbase0->US_RHR;
        else
                Zeichen = 0;

        return Zeichen;
}
```

```
@-------------------------------------------------------------------------
@ File Name:        ser_io.S
@ Object:           Ein- Ausgabe-Funktionen der seriellen Schnittstelle
@                   welche ueber den Supervisor-Mode gehen
@
@ Namen :                   Matr.-Nr.:
@
@-------------------------------------------------------------------------
@ Debuginformationen
        .file       "ser_io.S"
@ Funktion
        .text
        .align      2
        .global     initsswi
        .type       initsswi,function
initsswi:
        swi         0x100
        bx          lr          @ Rücksprung
@ Funktion
        .text
        .align      2
        .global     putcswi
        .type       putcswi,function
putcswi:
        mov         r1, r0      @ Zeichen nach r1
        ldr         r0, =Zeichen        @ Adresse der globalen Variablen holen
        str         r1, [r0]    @ Zeichen in globale Variable
        swi         0x200       @
        ldr         r1, =Zeichen        @ Adresse der globalen Variablen holen
        ldr         r0, [r1]    @ Zeichen aus globalen Variable
        bx          lr
@ Funktion
        .text
        .align      2
        .global     getcswi
        .type       getcswi,function
getcswi:
        ldr         r0, =Zeichen        @ Adresse der globalen Variablen holen
        swi         0x300
        ldr         r0, =Zeichen        @ Adresse der globalen Variablen holen
        ldr         r0, [r0]    @ empfangenes Zeichen zurueck geben
        bx          lr
@ Funktion
        .text
        .align      2
        .global     putsswi
        .type       putsswi,function
putsswi:
        stmfd sp!,{lr}          @ Retten der Register

// Hier koennte Ihr Code eingefügt werden.

        ldmfd sp!,{pc}          @ Rücksprung
@ Funktion
        .text
        .align      2
        .global     getsswi
        .type       getsswi,function
getsswi:
        stmfd       sp!,{lr}    @ Retten der Register

// Hier könnte Ihr Code eingefügt werden!

        ldmfd       sp!,{pc}    @ Rücksprung

        .data
Zeichen:  .byte 0
.end
```

```
/*---------------------------------------------------------------------------
@ File Name:        swi.c
@ Object:           SoftwareInterruptHandler
@
@ Autor:            Horsch/Pester
@ Datum:            3.12.2007/Januar2011
@---------------------------------------------------------------------------*/
void SWIHandler () __attribute__ ((interrupt ("SWI")));

void SWIHandler()
{
        register int reg_r0 asm ("r0");
        register int *reg_14 asm ("r14");

        switch( *(reg_14 - 1) & 0x00FFFFFF)     // Maskieren der unteren 24 Bits

                                                // und Verzweigen in Abh. der SWI Nummer
        {
                case 0x100:
                        init_ser();
                        break;
                case 0x200:
                        *((char *)reg_r0) = putch(*((char *)reg_r0));
                        break;
                case 0x300:
                        *((char *)reg_r0) = (unsigned int) getch();
                        break;
        }
}
```