h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

WS2024

Termin 5

APCS, Logische und Arithmetische Funktionen auf einem Zielsystem

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

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

Ziele:
Festigen der Kenntnisse über die APCS. Assemblerprogramme mit möglichst geringer Codegröße umzusetzen, sowie der Umgang mit Debugger, Simulator und dem Testen auf einem realen Zielsystem.

Arbeitsverzeichnis:
Kopieren Sie sich das Verzeichnis, welches Ihnen im Praktikum zur Verfügung gestellt wird, in Ihr persönliches Arbeitsverzeichnis. Dort stehen Ihnen dann alle benötigten Dateien, um die Aufgabenstellung lösen zu können, zur Verfügung.

Vorbereitung
Arbeiten Sie sich in folgende Befehle des ARM-Prozessors und in den ARM Procedure Call Standard (APCS) ein:

Instruktion	Bedeutung
STMDB R13, {R4-R8, LR}	Speichert die Registerwerte R4 bis R8 sowie LR (=R14) an die Adresse, die in R13 (=SP) steht als voll absteigender Stack
LDMIA R13, {R4-R8, PC}	Lädt den Speicherinhalt von der Adresse, die in R13 (=SP) steht in Form eines voll absteigenden Stacks in die Register R4 bis R8 sowie PC (=R15)

Aufgabe 1:
In folgenden Tabellen ist jeweils ein Speicherauszug gezeigt.

Welche Werte stehen in den Registern; auf den Speicherstellen nach Ausführung der Blocktransferbefehle?

R9 = 0x8000

Inhalt	Adresse
11	0x8014
10	0x8010
9	0x800C
8	0x8008
7	0x8004
6	0x8000
5	0x7FFC
4	0x7FF8
3	0x7FF4
2	0x7FF0

LDMDA R9, {R1, R2, R6}

R1:

R2:

R6:

R9:

STMIB R9!, {R1, R2, R6}

R1:

R2:

R6:

R9:

Inhalt	Adresse
	0x8014
	0x8010
	0x800C
	0x8008
	0x8004
	0x8000
	0x7FFC
	0x7FF8
	0x7FF4
	0x7FF0

Aufgabe 2:
Schreiben Sie ein beliebiges, kleines Programm in ARM Assembler, das durch Unterprogramme strukturiert wird. Folgende Anforderungen werden an das Programm und die Unterprogramme gestellt:

Die APCS Konvention ist einzuhalten.

Das Hauptprogramm soll (mindestens) drei Unterprogramme (UPx) aufrufen.

- UP1 benutzt Nicht-Scratchregister und stellt keine Blattroutine dar (ruft somit weitere Unterprogramme auf)
- UP2 benutzt nur Scratchregister und stellt eine Blattroutine dar (ruft somit keine weiteren Unterprogramme auf)
- UP3 benutzt nur Scratchregister und stellt keine Blattroutine dar (ruft z.B. UP1 auf)

Aufgabe 3:

Entwickeln Sie den benötigten Assemblercode für folgende Funktionen:

DS1 = SW1 AND SW2

DS2 = SW1 OR SW2

DS3 = SW1 EOR SW2

DS4/DS5 = SW1 + SW2

DS6/DS7 = SW1 – SW2

Gegeben

- Ist ein C-Programm *Termin5Aufgabe1.c* welches die Hardware initialisiert, alle Leuchtdioden DS1 (Bit 8) bis DS8 (Bit15) ausschaltet und die Tasten zur Abfrage vorbereitet. Es wird eine in Assembler zu schreibende Funktion

  „int Operationen(unsigned int* u_int_Taster, unsigned int* u_int_LedsOn)"

  aufgerufen welche die Leuchtdioden DS1 bis DS8 entsprechend der geforderten logischen und arithmetischen Funktionen einschaltet und die Information über den Zustand der Taste SW3 (Bit 5) liefert.
  Bei gedrückter Taste SW3 sollen die Leuchtdioden DS1 bis DS8 ausgeschaltet werden.
- Ist ein Assemblerprogrammgerüst *Funktionen.S* welches Sie zuerst vervollständigen und testen sollen, um es dann mit den verlangten Funktionen zu ergänzen.

Aufgabe 4

Machen Sie sich mit dem Gegebenen vertraut. Bringen Sie das gegebene Assemblerprogramm *Funktionen.S* in eine APCS-konforme funktionsfähige Form. Wie verhält sich das Programm?

Aufgabe 5

Lesen Sie in das Register R2 den Inhalt des Pin Data Status Register (PDSR) ein. Adresse vom PDSR wird in R0 übergeben. Testen und analysieren Sie die Informationen (Bit's) die für die Lösung der weiteren Aufgaben notwendig sind.

Aufgabe 6

Schalten Sie die LED's DS1 bis DS8 (Low-Aktiv) gezielt ein. Die Adresse für das Clear Output Data Register (CODR) wird in Register R1 übergeben.

Aufgabe 7

Ergänzen Sie das Assemblerprogramm nun so, dass die geforderte Funktionalität gegeben ist. Testen Sie ob die Leuchtdioden DS1 bis DS7 beim Drücken der Tasten SW1 und SW2 die richtigen Ergebnisse zeigen.

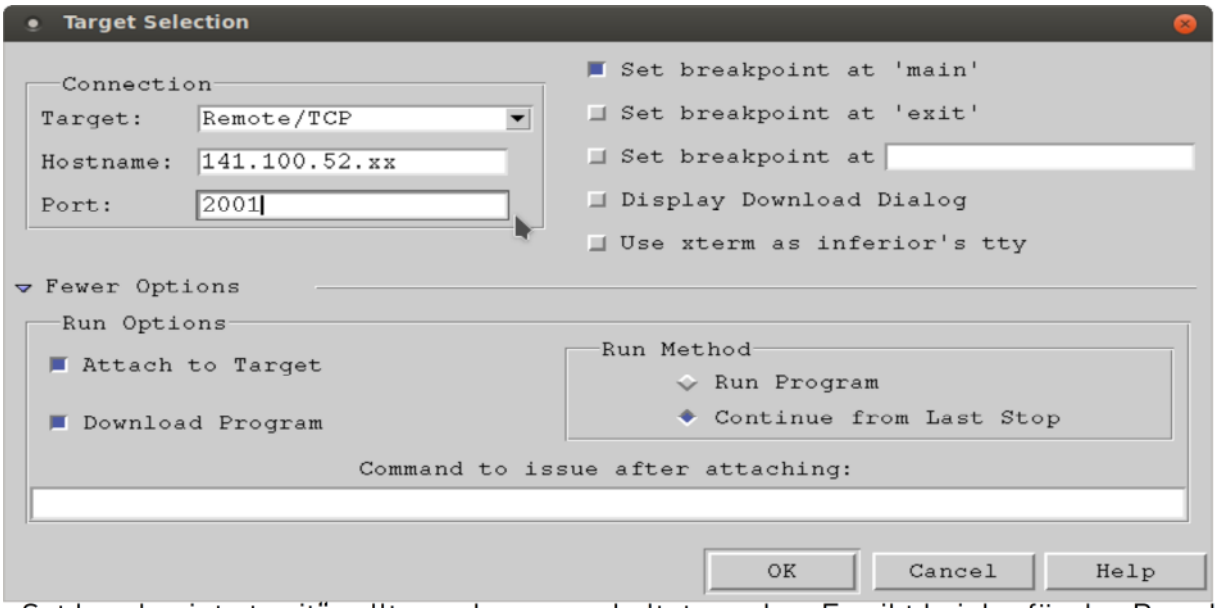
Aufgabe 8

Das Programm soll nun noch, abhängig vom Zustand der Taste SW3, ein TRUE oder FALSE in Register R0 zurück geben.

Informationen

Sie können die Entwicklung und erste Tests auch im Simulator erledigen. Es ist natürlich etwas mühsam/umständlich die Tastendrücke KEY1 bis KEY3 durch Manipulation der Registerinhalte zu simulieren. Statt sich mit dem Target Simulator zu verbinden, können Sie im Mikroprozessorlabor auch die ARM7-Evaluationssysteme (AT91EB63) nutzen. Hierzu schalten sie Steckdosenleiste auf Ihrem zum Arbeitsplatz gehörenden System an. Nachdem sich das System mit dem auf Ihrem

Arbeitsplatzrechner laufenden TFTP-Server verbunden hat (erkenntlich durch Leuchten der LED's DS6 bis DS8), können Sie sich aus dem Debugger mit dem Target Remote/TCP über den BDI2000 mit dem System (AT91EB63) verbinden. Wählen Sie für den Hostname die IP-Adresse, welche auf dem zugehörigen BDI2000 steht. Als Port wählen Sie die 2001. Weitere Einstellungen entnehmen Sie der folgenden Abbildung.



„Set breakpoint at exit" sollte noch ausgeschaltet werden. Es gibt bei der für das Board gebauten Software kein Label „exit".

Das zur Verfügung gestellte *makefile* ist schon auf die Nutzung des Testsystem ausgelegt. Es werden das Programm und die Daten im externen RAM (ab Adresse 0x2000000) des Testsystems abgelegt.

```
// Gegebenes C-Programm zur Lösung von Termin5 ab  Aufgabe 3
// von: Manfred Pester
// vom:  18.02.2013 letzte Aenderung: 07.09.2023

#include "../h/pmc.h"
#include "../h/pio.h"

// Die Funktion „int f_int_Operationen(unsigned int* u_int_Taster, unsigned int* u_int_LedsOn)"
// erwartet die Registeradresse fuer das Pin Data Status Register PDSR aus dem die
// Tasteninformationen SW1 bis SW3 gelesen werden koennen
// und die Registeradresse fuer das Clear Output Data Register CODR um die Leuchtdioden DS1 bis DS8 einschalten zu koennen
// und liefert TRUE oder FALSE  zurueck.

int f_int_Operationen(unsigned int* u_int_Taster, unsigned int* u_int_LedsOn);

int main(void)
{
        int int_Weiter = 1;
//
//************************************************

// PowerManagementController Clock fuer PIOB einschalten
        PMC_BASE->PMC_PCER = 0x4000;

// Parallel I/O Controller PIOB Taster und Leuchtdioden aktivieren
        PIOB_BASE->PIO_PER = 0xff38;        // alle Leuchtdioden DS1..DS8 und Taster SW1..SW3 aktiv
        PIOB_BASE->PIO_OER = 0xff00;        // fuer alle Leuchtdioden DS1..DS8 Ausgaenge aktivieren

// über das Register PIOB_BASE->PIO_PDSR können die Tasten-Bits gelesen werden
// über das Register PIOB_BASE->PIO_CODR können LED-Bits gelöscht werden (cleared via Maske)
// über das Register PIOB_BASE->PIO_SODR können LED-Bits gesetzt werden (set via Maske)
// siehe auch S. 66 der im Labor ausliegenden Dokumentation für den ATMEL AT91
        while(1)
        {
                int_Weiter = f_int_Operationen(&PIOB_BASE->PIO_PDSR, &PIOB_BASE->PIO_CODR);
                if (int_Weiter)
                        PIOB_BASE->PIO_SODR = 0xff00;        // alle 8 LED aus (lowaktiv)
        }
        return 0;
}
```

Gegebenes Assemblerprogrammgerüst:

```
@ int f_int_Operationen(unsigned int* u_int_Taster,  unsigned int* u_int_LedsOn)
@ Diese Funktion soll auf
@ - LED DS1 (Bit 8) das Ergebnis von SW1 (Bit 3) AND SW2 (Bit 4) anzeigen
@ - LED DS2 (Bit 9) das Ergebnis von SW1 OR SW2 anzeigen
@ - LED DS3 (Bit 10) das Ergebnis von SW1 EOR SW2 anzeigen
@ - LED DS4 und 5 (Bit 11/12) das Ergebnis von SW1 ADD SW2 anzeigen
@ - LED DS6 und 7 (Bit 13/14) das Ergebnis von SW1 SUB SW2 anzeigen
@ und bei gedrueckter TASTE SW3 (Bit 5) soll die Funktion ein TRUE (Wert ungleich 0) ansonsten FALSE (Wert gleich 0) an das
@ aufrufende Programm zurueck geben.
@ ACHTUNG die Tasten und auch die Leuchtdioden sind Low-Aktiv beschaltet.
        .file   "Funktionen.S"
        .text
        .align  2
        .global f_int_ Operationen
        .type   f_int_Operationen, %function
f_int_Operationen:

// AND

// OR

// EOR

// ADDITION

// SUBTRAKTION


// Beenden
        bx      lr

.Lfe1:
        .size   f_int_Operationen,.Lfe1- f_int_ Operationen
// End of File
```