MIKROPROZESSORPRAKTIKUM

WS2018

Termin1

C-Programmierung für eingebettete Systeme

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

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

Lernziele:

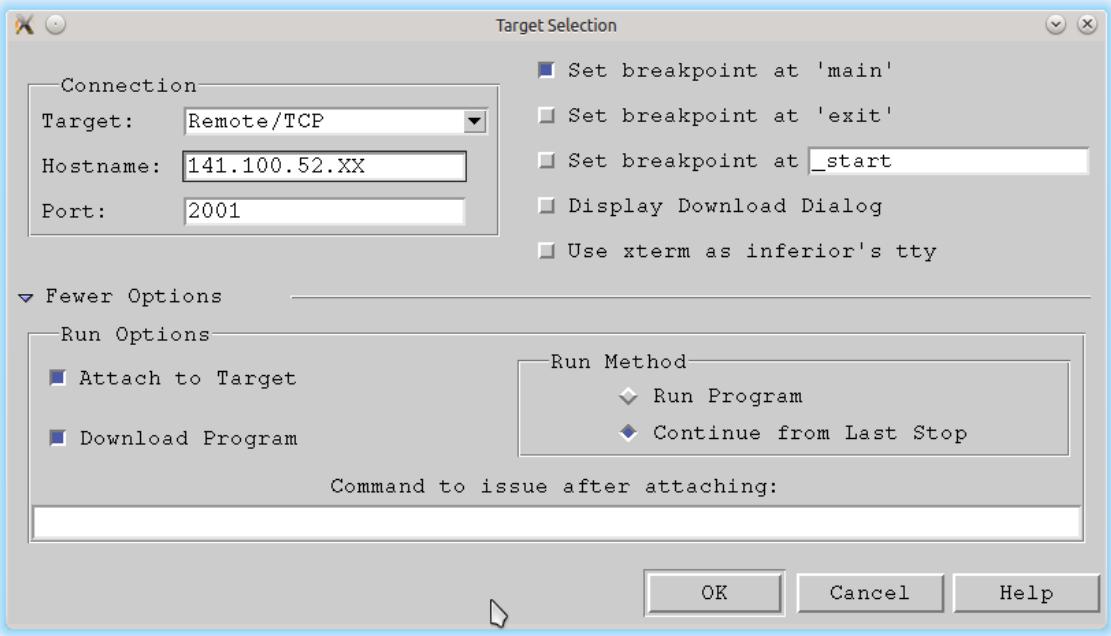
Mit den folgenden Versuchen sollen Sie die Sprache "C" einmal aus einer anderen Sicht kennen lernen. An ganz einfachen Programmen sollen Sie ermitteln, welchen Code ein Compiler erzeugt, wo welche Variablen abgelegt werden, welchen Einfluss die Optimierungsstufen haben, wie ein c*all by value / reference* in ARM Assembler umgesetzt wird. Wie auf Peripherie (System on Chip) zugegriffen wird.

Arbeitsverzeichnis:

Kopieren Sie sich das Verzeichnis /mpsWS2018. Dort stehen im Unterverzeichnis Termin1 die Dateien *Termin1AufgabeX.c als* Programmgerüstbeispiele und *makefile* zur Verfügung.

Infos

Es ist natürlich etwas mühsam/umständlich die nicht vorhandenen LED DS1-DS8, Tastendrücke KEY1 bis KEY3 durch sichten und manipulieren 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 die Steckdosenleiste an Ihrem zum Arbeitsplatz gehörenden System an. Nachdem sich das System mit dem auf Ihrem Arbeitsplatzrechner laufenden TFTP-Server verbunden hat, können Sie sich aus dem Debugger mit dem Target Remote/TCP über den BDI2000 mit dem System (AT91EB63) verbinden. Die Verbindung und die Initialisierung hat funktioniert, wenn aus dem Lauflicht der Dioden ein statisches Leuchten der rechten 3 LED (DS6 bis DS8) wurde. 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.



Aufgabe 1:

Legen Sie im C Programm zwei lokale integer Variablen an. Weisen Sie diesen im Code Werte zu, z.B. 0x1 und 0x2. Übersetzen Sie Ihr Programm und schauen Sie sich den erzeugten Code an. Führen Sie das Programm im Debugger aus. Dokumentieren Sie den erzeugten Code.

Hinterfragen Sie Ihre Beobachtungen z. B. mit:
Wo liegen die Werte der Variablen im Speicher?
Wie kommen die Werte in den Speicher?
Welche Register werden verwendet?
..

Ändern Sie im Makefile die Optimierungsstufe und beobachten und dokumentieren Sie die Veränderungen.

Aufgabe 2:

Verwenden Sie nun statt der lokalen Variablen globale Variablen.
Was ändert sich?
Warum?
Wo liegen die Werte der Variablen im Speicher?
..

Aufgabe 3:

Legen Sie nun zwei globale und zwei lokale integer Variablen an. Weisen Sie den Variablen Werte zu. Machen Sie nun Zuweisungen der Form „global=lokal" und/oder „lokal=global".
Was stellen Sie fest?
Ändern Sie im Makefile die Optimierung und beobachten und dokumentieren Sie die Auswirkungen.
..

Aufgabe 4:

Vereinbaren Sie einen Funktionsprototypen für eine Funktion "int addition(int, int, int)", die 3 integer Parameter erwartet. Rufen Sie diese Funktion im Anschluss an die Zuweisung nach Aufgabe 2 und 3 mit den beiden lokalen und einer globalen Variablen auf. Dokumentieren Sie Ihre Beobachtungen. Ändern Sie im Makefile die Optimierung und beobachten und dokumentieren Sie die Auswirkungen.

Aufgabe 5:

Wir werden uns nun mit den Registern der PIO des hier eingesetzten Mikrocontroller (siehe auch Doku AT91M6320.pdf) beschäftigen.

Name	Adresse	Bedeutung
PIOB_PER	0xFFFF0000	PIOB Port Enable Register
PIOB_OER	0xFFFF0010	PIOB Output Enable Register
PIOB_SODR	0xFFFF0030	PIOB Set Output Data Register
PIOB_CODR	0xFFFF0034	PIOB Clear Output Data Register

Schreiben Sie zunächst den Wert 0x100 ins PIOB_PER und dann ins PIOB_OER. Danach schreiben Sie nacheinander den Wert 0x100 einige Male alternierend ins PIOB_SODR und PIOB_CODR. Dadurch sollte die LED DS1 auf dem Board AT91EB63 an und aus gehen. Testen Sie die Funktion im Debugger mit Einzelschritten aus. Im gegebenen Programmbeispiel Termin1Aufgabe5.c sind verschiedene Möglichkeiten des Beschreibens der Register der PIO gezeigt. Beschäftigen Sie sich mit den verschiedenen Programmiermöglichkeiten.

Welche Variante würden Sie bevorzugen und warum?

Aufgabe 6:

Versuchen Sie nun Ihr Programm auf minimale Programmgröße zu bringen.
Wieviele Byte Speicher benötigen Sie für die Initialisierung und um die LED in einer Endlosschleife blinken zu lassen? Aus wie vielen Assemblerbefehlen besteht Ihr minimiertes Programm?

Aufgabe 7:

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 auch daran, dass Sie Programmteile in den nächsten Praktika nochmals benötigen. Beschäftigen Sie sich baldigst auch mit den Aufgaben der nächsten Termine.

Beispiel eines makefile zu Termin1 Aufgabe1

```
# zu verwendete Quelldatei
FILE = Termin1Aufgabe1
# Toolchain
TOOLCHAIN = arm-elf-
# Compiler
COMPILER = gcc
# Linker/Binder
LINKER = ld
# Debugger
DEBUGGER = insight
# Optimierungsstufen
OPTI = 0

# Bauen
all:
# uebersetzen der Quelldatei (FILE)
# Der Schalter -c erzeugt nur die Objektdatei aus der Quelldatei ohne zu binden
# Der Schalter -g in gcc fügt Debugging-Code in die kompilierten Objektdateien ein
# Der Schalter -O gibt die zu verwendete Optimierungsstufe (0..3,s) an
# Der Schalter -I weist gcc an, das Verzeichnis include in den Include-Pfad einzufügen
# Der Schalter -L weist gcc an, das Verzeichnis lib in den Library-Pfad einzutragen.

	$(TOOLCHAIN)$(COMPILER) -c -g -O$(OPTI) $(FILE).c -I ../h

# Der Schalter -S erzeugt eine Assemblerdatei aus der Quelldatei
	$(TOOLCHAIN)$(COMPILER) -S -O$(OPTI) $(FILE).c

# Erzeugen weitere benoetitgen Objektdateien
	$(TOOLCHAIN)$(COMPILER) -c -g -O$(OPTI) ../boot/swi.S -o swi.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 boot_ice.o swi.o $(FILE).o -o $(FILE).elf

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

# Aufraeumen
clean:
	rm *.o
	rm *.elf
```