Embedded Systems

Assembler

Praktikum 5

Fachhochschule Bielefeld
Campus Minden
Studiengang Informatik

Beteiligte Personen:

Name	Matrikelnummer
Jan-Hendrik Sünderkamp	1153536
Peter Dick	1050185

---

3. Juni 2018

Inhaltsverzeichnis

1 Vorbereitung 3

2 Aufgabe 1 3

3 Aufgabe 2 7

4 Aufgabe 3 8

1 Vorbereitung

Infomaterial finden Sie u.a. im Ilias:

- ARMv7-M Architecture Reference Manual, Kapitel A4 (”The ARMv7-M Instruction Set”) und A5 (”Thumb Instruction Set Encoding”)
- Cortex-M4 Devices Generic User Guide, Kapitel 3 ”The Cortex-M4 Instruction Set”

2 Aufgabe 1

Sehen Sie sich an, wofür die Optionen *-O0*, *-O2* und *-Os* des Compilers *arm-none-eabi-gcc* verwendet werden, z.B. `https://gcc.gnu.org/onlinedocs/gcc-6.4.0/gcc/Optimize-Options.html`.

- -O0: Verringert die Kompilierungszeit und macht das beim debuggen die erwarteten Ergebnisse produziert werden.
- -O2: GCC führt nahezu alle unterstützen Optimierungen aus die nicht einen Speicher-Geschwindigkeit-Kompromiss eingehen. Diese Option erhöht die Kompilierungszeit und die Performance des generierten Code.
- -Os: Optimiert die Code-Größe.

Setzen Sie den Pfad für ausführbare Programme auf die Tools der (TivaC-)Toolchain der Energia-Entwicklungsumgebung und öffnen Sie ein Fenster mit einer Kommandozeile. Ermitteln Sie Ihre Comiler Version (*arm-none-eabi-gcc –version*). Idealerweise haben Sie die Version 6.3.1.

Der Pfad kann unter Systemsteuerung/System und Sicherheit/System/Erweiterte Systemeinstellungen/Erweitert/Umgebungsvariablen gesetzt werden.

Laden Sie die vier Dateien *blink.c*, *startup_gcc.c*, *EK-TM4C123GXL.ccxml* und *blink.ld* aus ILIAS. Für diese Aufgabe betrachten wir hauptsächlich die Datei blink.c (s. Listig).

Übersetzen Sie die Dateien und bauen Sie eine ELF-Datei für das TivaC-Board mit Optimierung *-O0*. Ermitteln Sie die Programmgröße und generieren Sie ein Disassembly. Sie können folgende Befehle in der Kommandozeile ausführen (oder sich ein Makefile schreiben). Für die folgenden Befehle ist der *PATH_TO_SYSTEM_DIR* durch den Pfad zum TivaC ”system”-Ordner in ihrer Energia-Installation zu ersetzen, z.B. ”C:\Users\User\AppData\Local\Energia15\packages\energia\hardware\tivac\1.0.3\system”:

arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -O0 -IPATH_TO_SYSTEM_DIR -std=gnu11 -c -o blink.o blink.c

arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -O0 -IPATH_TO_SYSTEM_DIR -std=gnu11 -c -o startup_gcc.o startup_gcc.c

arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -O0 -T blink.ld -Xlinker –gc-sections -o blink.elf blink.o startup_gcc.o

arm-none-eabi-size blink.elf

arm-none-eabi-objdump -h -S blink.elf > blink0.asm

```
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"

 uint32_t ui32Loop;

void delay() {
    for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++);
}

int main(void) {
    // Enable the GPIO port.
    SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF;

    // Do a dummy read to insert a few cycles
    ui32Loop = SYSCTL_RCGC2_R;

    // Set up GPIO port.
    GPIO_PORTF_DIR_R = 0x08;
    GPIO_PORTF_DEN_R = 0x08;

    // Loop forever.
    while(1) {
        // Turn on the LED.
        GPIO_PORTF_DATA_R |= 0x08;

        // Delay for a bit.
        delay();

        // Turn off the LED.
        GPIO_PORTF_DATA_R &= ~(0x08);

        // Delay for a bit.
        delay();
    }
}
```

Listing der Datei blink.c

Sie können das Programm mit den folgenden Schritten auch auf das Board überspielen:
DSLite load -c EK-TM4C123GXL.ccxml blink.elf

Sie sollten nun die Größe des Programmcodes ermittelt und u.a. die Datei *blink0.asm* generiert haben. In der Datei *blink0.asm* finden Sie ein Disassembly des Programms *blink.c*.

Die Größe des Programmcodes beträgt 1148 bytes.

Gehen Sie zu der 10. Zeile nach der Zeile mit *000029c <main>*: Die Zeile hat die folgende Form (erste drei Zeichen ggf. anders):
2ae: *4b0d ldr r3, [pc, #52] ; (2e4 <main+0x48>)*

Erklären Sie, was Ihnen das Disassembly in den Spalten zeigt. Erklären Sie weiterhin genau die Anweisungen der Assemblerbefehle dieser und der folgenden 18 Zeilen bis zur Zeile mit
2da: *e7ee b.n 2ba <main+0x1e>*

In der ersten Spalte steht die Adresse des Assemblerbefehls.
In der zweiten Spalte steht der Assemblerbefehl in Hexadezimale Darstellung.
In der dritten Spalte steht der Assemblerbefehl in "Menschen Lesbaren Darstellung" z. B. add.
In der vierten Spalte stehen die Operanden die zu dem Assemblerbefehl (Spalte 3) gehören.
In der fünften Spalte steht ein Kommentar mit weiteren Informationen z. B. die Adresse des zu lesendes Wortes.

2ae: ldr r3, [pc, #52] ; (2e4 <main+0x48>)
Der pc-Register zeigt auf die Adresse des nächsten Befehls also 0x2b0.
"ldr r3, [pc, #52]" schreibt den Inhalt der angegebenen Adresse (Adr. im Reg. pc + #52 = 0x2b0 + 0x34 = 0x2e4) in den Register r3. Das Wort 40025400 wird in den Register r3 geschrieben.

2b0: movs r2, #8
"movs r2, #8" schreibt den Wert #8 in den Register r2.

2b2: str r2, [r3, #0]
"str r2, [r3, #0]" schreibt den Inhalt des Registers r2 in die Angegebene Adresse (Adr. im Reg. r3 + #0 = 0x40025400 + 0x0 = 0x40025400). Das heißt der Wert #8 aus Register r2 wird in die Adresse 0x40025400 geschrieben.

2b4: ldr r3, [pc, #48] ; (2e8 <main+0x4c>)
"ldr r3, [pc, #48]" schreibt den Inhalt der angegebenen Adresse (Adr. im Reg. pc + #48 = 0x2b8 + 0x30 = 0x2e8) in den Register r3. Das Wort 4002551c wird in den Register r3 geschrieben.

2b6: movs r2, #8
"movs r2, #8" schreibt den Wert #8 in den Register r2.

2b8: str r2, [r3, #0]
"str r2, [r3, #0]" schreibt den Inhalt des Registers r2 in die Angegebene Adresse (Adr. im Reg. r3 + #0 = 0x4002551c + 0x0 = 0x4002551c). Das heißt der Wert #8 aus Register r2 wird in die Adresse 0x4002551c geschrieben.

2ba: ldr r2 [pc, #48] ; (2ec <main+0x50>)
"ldr r2 [pc, #48]" schreibt den Inhalt der angegebenen Adresse (Adr. im Reg. pc + #48 = 0x2bc + 0x30 = 0x2ec) in den Register r2. Das Wort 400253fc wird in den Register r2 geschrieben.

2bc: ldr r3 [pc, #44] ; (2ec <main+0x50>)
"ldr r3 [pc, #44]" schreibt den Inhalt der angegebenen Adresse (Adr. im Reg. pc + #44 = 0x2c0 + 0x2c = 0x2ec) in den Register r3. Das Wort 400253fc wird in den Register r3 geschrieben.

2be: ldr r3 [r3, #0]
"ldr r3 [r3, #0]" schreibt den Inhalt der angegebenen Adresse (Adr. im Reg. r3 + #0 = 0x400253fc + 0x0 = 0x400253fc) in den Register r3.

2c0: orr.w r3, r3, #8
"orr.w r3, r3, #8" schreibt das Ergebnis einer bitweisen ODER-Operation mit den Operatoren r3 und #8 in Register r3 (r3 = r3 | #8).

2c4: str r3, [r2, #0]
"str r3, [r2, #0]" schreibt den Inhalt des Registers r3 in die Angegebene Adresse (Adr. im Reg. r2 + #0 = 0x400253fc + 0x0 = 0x400253fc). Das heißt der Wert aus Register r3 wird in die Adresse 0x400253fc geschrieben.

2c6: bl 26c <delay>
"bl 26c <delay>" springt zur Adresse 0x26c (Funktion delay).

2ca: ldr r2 [pc, #32] ; (2ec <main+0x50>)
"ldr r2 [pc, #32]" schreibt den Inhalt der angegebenen Adresse (Adr. im Reg. pc + #32 = 0x2cc + 0x20 = 0x2ec) in den Register r2. Das Wort 400253fc wird in den Register r2 geschrieben.

2cc: ldr r3 [pc, #28] ; (2ec <main+0x50>)
"ldr r3 [pc, #28]" schreibt den Inhalt der angegebenen Adresse (Adr. im Reg. pc + #28 = 0x2d0 + 0x1c = 0x2ec) in den Register r3. Das Wort 400253fc wird in den Register r3 geschrieben.

2ce: ldr r3 [r3, #0]
"ldr r3 [r3, #0]" schreibt den Inhalt der angegebenen Adresse (Adr. im Reg. r3 + #0 = 0x400253fc + 0x0 = 0x400253fc) in den Register r3.

2d0: bic. r3, r3, #8
"bic. r3, r3, #8" schreibt das Ergebnis einer bitweisen AND-Operation mit den Operator r3 und den bitweise negierten Operator #8 in den Register r3 (r3 = r3 & (#8)).

2d4: str r3, [r2, #0]
"str r3, [r2, #0]" schreibt den Inhalt des Registers r3 in die Angegebene Adresse (Adr. im Reg. r2 + #0 = 0x400253fc + 0x0 = 0x400253fc). Das heißt der Wert aus Register r3 wird in die Adresse 0x400253fc geschrieben.

2d6: bl 26c <delay>
"bl 26c <delay>" springt zur Adresse 0x26c (Funktion delay).

2da: b.n 2ba <main +0x1e>
"b.n 2ba <main +0x1e>" springt zur Adresse 0x2ba (While true schleife in main).

3 Aufgabe 2

Führen Sie nun die oben dargestellten Schritte mit der Option *-O2* aus. Generieren Sie ein weiteres Disassembly in einer Datei *blink2.asm*.

Vergleichen Sie die beiden Größen der Programmcodes, die mit den unterschiedlichen Optimierungsparametern generiert wurden. Vergleichn Sie auch den Inhalt der Dateien *blink0.asm* und *blink2.asm*. Welche wesentliche Änderung hat der Compiler am auszufürenden Binärcode (siehe die Disassemblies *.asm*) durchgeführt?

Die Größe des Programmcodes (blink0.asm) beträgt 1148 bytes.
Die Größe des Programmcodes (blink2.asm) beträgt 1104 bytes.
Der Programmcode von blink2.asm ist 44 bytes kleiner.
Die Funktion delay() ist in der main() Funktion integriert.
Es werden die Register r0-r6 verwendet (in der Main Funktion).
Die Wörter werden nur einmal in der Main geladen und nur noch mithilfe von Registern geladen und gespeichert.

4 Aufgabe 3

Ändern Sie nun das Programm, indem Sie das Keyword *volatile* im Quellcode *blink.c* löschen. Führen Sie die oben beschriebenen Schritte mit der Option *-O2* nochmals durch und sehen Sie Sich das neue Disassembly an.

Was und warum hat der Compiler nun etwas Anderes generiert? Wie wirkt sich dies auf die Programmausführung auf dem Board aus?

Der Compiler lässt durch das ”volatile”-Keyword einige Optimierungen aus. Dem Entsprechend ist der neue Code um 16 Zeilen kürzer (gemessen bis zu b.n-Zeile) und weißt die Operationen ”cmp”, ”adds”, ”bls” und ”bhi” nicht mehr auf. Die Anzahl der genutzten Register ist gleich geblieben, lediglich die Namen haben sich geändert. Bei der Programmausführung auf dem Board werden allerdings erst die gravierenden Unterschiede durch die Optimierung klar.
Die Programmausführung **mit** dem Keyword sieht wie folgt aus:

1. Die LED des Boards ist aus.
2. Nach einem Knopfdruck beginnt die LED zu blinken.
3. Solange der Knopf gedrückt wird, ist die LED aus.

Ein Durchlauf **ohne** das Keyword sieht allerdings so aus:

1. Die LED des Boards ist an.
2. Solange der Knopf gedrückt wird, ist die LED aus.
3. Nach einem Knopfdruck leuchtet die LED wieder permanent.

Es ist also deutlich, dass in diesem Fall das ”volatile”-Keyword benötigt wird, damit das Programm funktioniert und nicht elementare Bestandteile weg-optimiert werden.