Praktikum 03 - Toolchain

Malte Riechmann, André Kirsch

Aufgabe 1

Zuerst haben wir die Energia Entwicklungsumgebung installiert. Danach haben wie über den Boardverwalter die Energia TivaC Boards installiert. Über das Internet haben wir die passenden Treiber für das TivaC Series Launchpad heruntergeladen. Nach dem Anschließen des Launchpads konnten wir die fehlenden Treiber über den Geräte Manager installieren.

Danach haben wir den Blink Sketch in die Energia Entwicklungsumgebung geladen und die geforderten Anpassungen gemacht.

```
#define LED RED_LED

const int delayTime = 4000;

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, HIGH);
  delay(delayTime);
  digitalWrite(LED, LOW);
  delay(delayTime);
}
```

Danach wurde der "verbose output" für die Kompilierung und den Upload eingeschaltet. Daraufhin wurde der Code geprüft und auf das Launchpad hochgeladen.

Aufgabe 2

2.1:

```
Detecting libraries used... //Sucht nach genutzten Bibliotheken
Generating function prototypes... //Generiert aus dem Sketch eine Datei
                                  //"ctags_target_for_gcc_minus_e.cpp
Sketch wird kompiliert... //Kompiliert den Sketch in eine .o-Datei
                         //(Präprozessing, Kompilierung, Assemblierung)
Compiling libraries...
Compiling core... //Kompiliert wie auch den Sketch den Energia Core und die Bibliotheken
Linking everything together... //Erstellt aus allem eine .elf Datei
                              //Aus der .elf Datei wird ein .bin Binärdatei
                              //Als letztes wird die .bin Datei hochgeladen
```

2.2:

Schritt	Tool
Präprozessing	arm-none-eabi-g++ / arm-none-eabi-gcc
Kompilierung	arm-none-eabi-g++ / arm-none-eabi-gcc
Assemblierung	arm-none-eabi-g++ / arm-none-eabi-gcc
Linken	arm-none-eabi-g++
Konvertierung	arm-none-eabi-objcopy
Hochladen	DSLite
Bibliothek erstellen	arm-none-eabi-ar

Aufgabe 3

Das Linker Script befindet sich unter
`C:\Users\Andre\AppData\Local\Energia15\packages\energia\hardware\tivac\1.0.3\variants\EK-TM4C123GXL`

Ein Linker Script beschreibt die Abbildung der Segmente (Memory Map) des Kompilats in den Speicher des Target-Systems:

- für Programmcode in ROM und RAM
- für Variablen im RAM
- für die Vektor-Interrupt-Tabelle

`Memory` Block:

```
MEMORY
{
    flash (rx) : ORIGIN = 0x00000000, LENGTH = 0x00040000
    ram  (rwx) : ORIGIN = 0x20000000, LENGTH = 0x00008000
}
```

Die einzelnen Zeilen im Memory Block beschreiben, welcher Speicherbereich wovon belegt wird.

Beispiel 1.Zeile:

Zeilenteil	Bedeutung
flash	Name des Speicherbereichs
(rx)	Lese- und Schreibrechte (in diesem Fall nur Leserechte)
ORIGIN = 0x00000000	Start des Speicherbereichs
LENGTH = 0x00040000	Länge des Speicherbereichs

Aufgabe 4

```
#define LED PB_3

const int longDelay = 1000;
const int shortDelay = 500;
const int delayBetweenSignal = 500;
const int delayBetweenChars = 1000;
const int delayBetweenWords = 2000;

void sendSignal(const int delay) {
  digitalWrite(LED, HIGH);
  delay(delay);
  digitalWrite(LED, LOW);
  delay(delayBetweenSignal);
}

void sendO() {
  for (int i = 0; i < 3; i++) {
    sendSignal(longDelay);
  }
  delay(delayBetweenChars);
}

void sendS() {
  for (int i = 0; i < 3; i++) {
    sendSignal(shortDelay);
  }
  delay(delayBetweenChars);
}

void sendSOS() {
  sendS();
  sendO();
  sendS();
  delay(delayBetweenWords);
}

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  sendSOS();
}
```

Aufgabe 5

Beide Dateien finden sich unter:
`C:\Users\Andre\AppData\Local\Energia15\packages\energia\hardware\tivac\1.0.3\cores\tivac`

Die main funktioniert so, dass zuerst die setup-Funktion aufgerufen wird und danach in einer Endlosschleife die loop-Funktion. Es reicht aus, die Funktionen setup und loop zu implementieren, da der Rest von der Energia Entwicklungsumgebung erledigt wird bzw. die main-Funktion diese setup-loop Implementierung bereits besitzt.